

Week0510.20230502-Grafana and InfluxDB

กลุ่มคำานณ _____ รหัส B6323059 ชื่อ-สกุล นายประชญา ธนาพิบูลผล

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/ทำความรู้กับ-grafana-dashboard-1a5efe6d170a>

4/4. การทดสอบ

- ให้ทำการทดสอบและเขียนขั้นตอนการทดสอบ โดยใช้ ESP32 ส่งข้อมูลไปยัง MQTT Broker และใช้ Grafana .
ในการมองนิเตอร์ข้อมูล 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/>

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

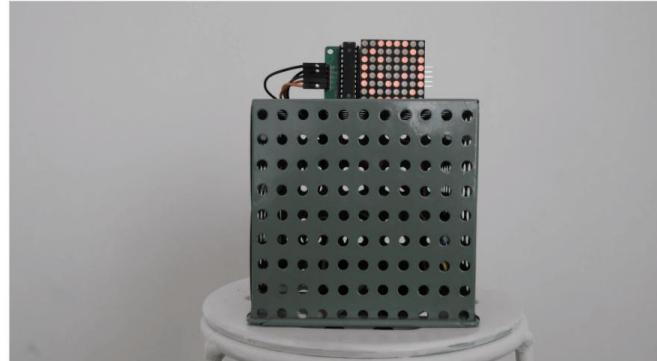
The hardware

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.

The screenshot shows the Grafana Cloud Portal interface. At the top, there's a header with the title "Grafana Cloud Portal", a "Contact Support" button, and a sub-header "Configure your Grafana Cloud resources and manage users, API keys, billing and more." Below the header, there are six service management cards arranged in two columns of three:

- Grafana**: Includes a "Log In" button and a "Details" button. Below it, it says "Log in to start using Grafana or view Details to manage Grafana Plugins." It also shows "Active Users: 0".
- Prometheus**: Includes a "Send Metrics" button and a "Details" button. Below it, it says "Set up and manage your Prometheus metrics service." It also shows "Active Series: 0".
- Loki**: Includes a "Send Logs" button and a "Details" button. Below it, it says "Set up and manage your Loki logging service." It also shows "Ingest Rate: 0 bytes/hr".
- Graphite**: Includes a "Send Metrics" button and a "Details" button. Below it, it says "Set up and manage your Graphite metrics service." It also shows "Active Series: 0".
- Alerts**: Includes a "Configure" button and a "Details" button. Below it, it says "Set up and manage your Prometheus-based alerts."
- Tempo**: Includes a "Send Traces" button and a "Details" button. Below it, it says "Set up and manage your Tempo tracing service."

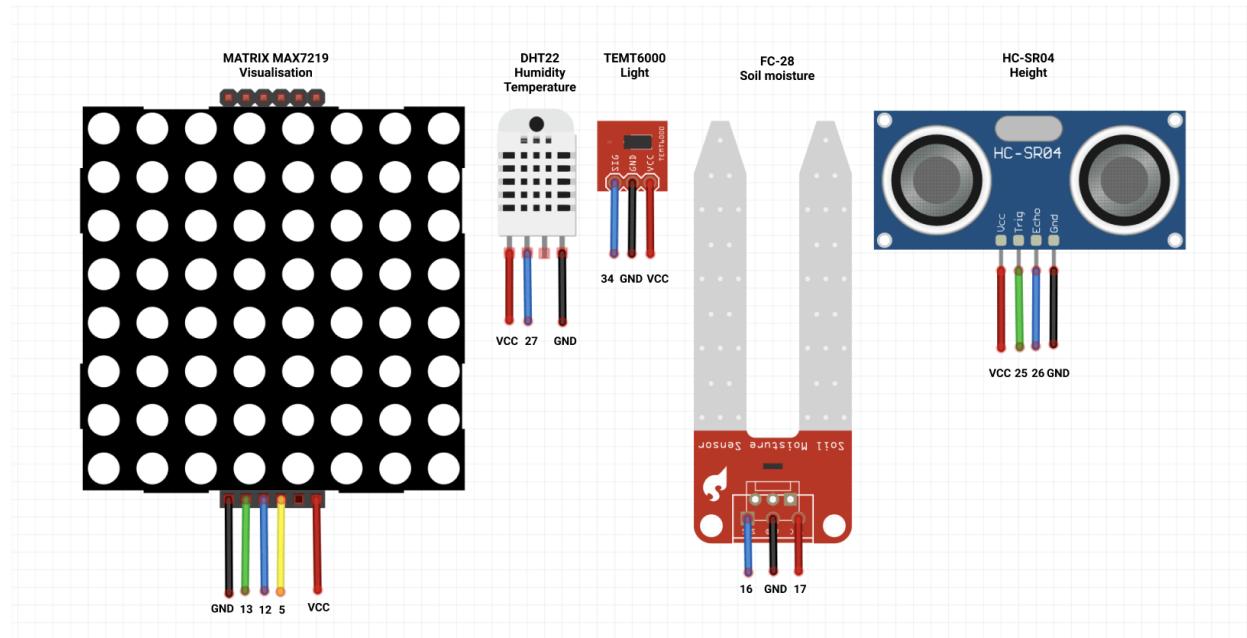
At this point, I've also created my API keys for Loki and Graphite to publish metrics from my monitoring system to these databases.

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

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

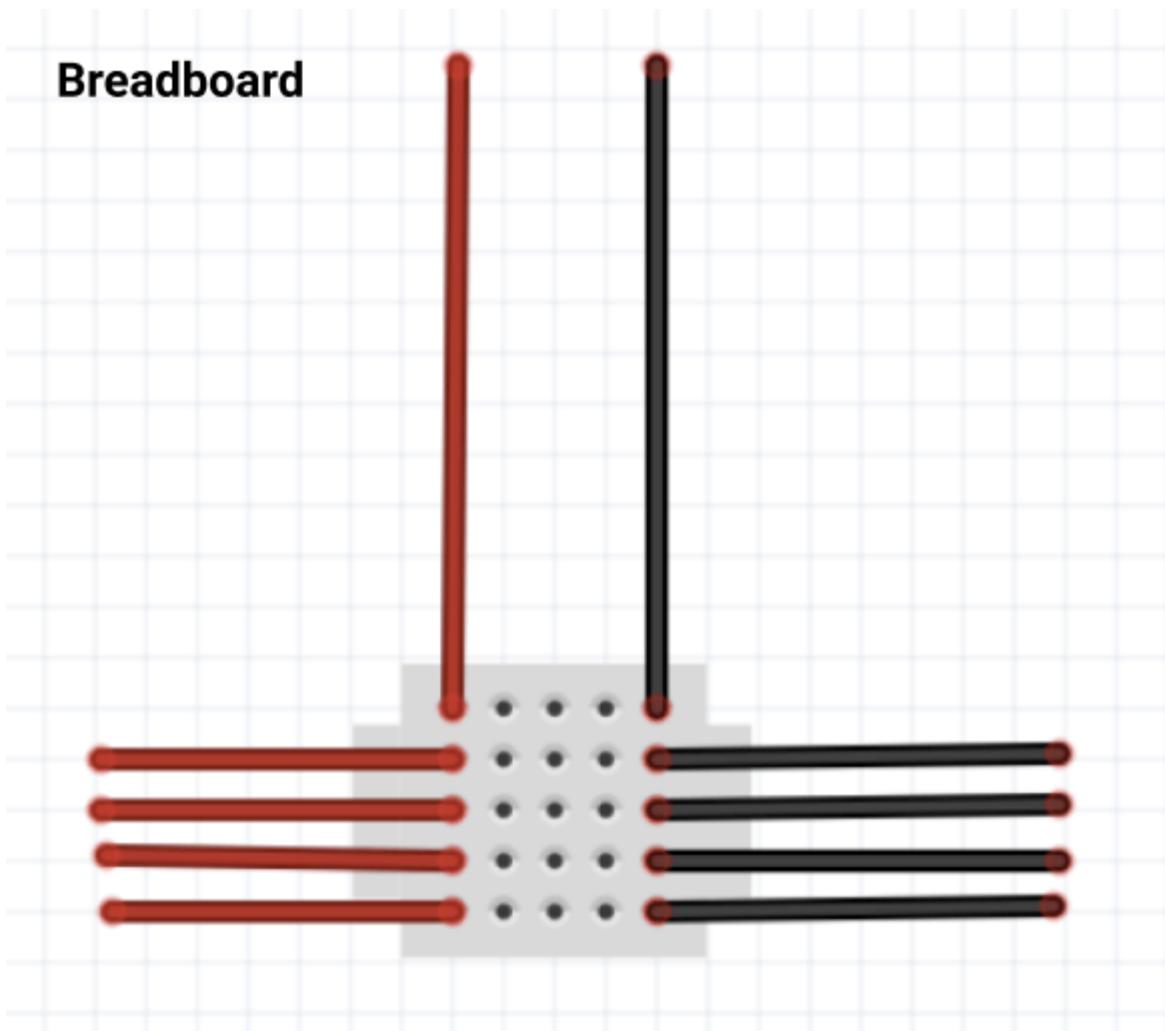
The hook-up

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:

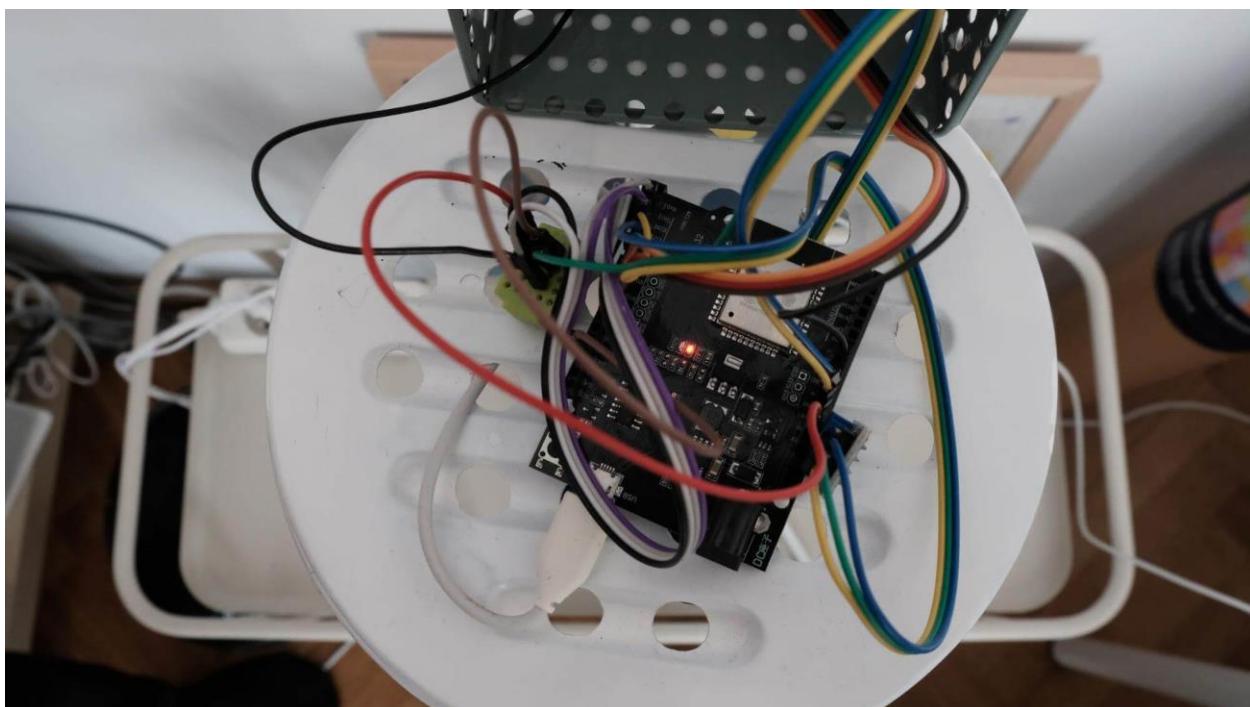
- VCC pin to pin 17 (I am not using the vcc pin to prevent corrosion)
- SIG output voltage pin to pin 16 on the board
- GND pin to GND on the board

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

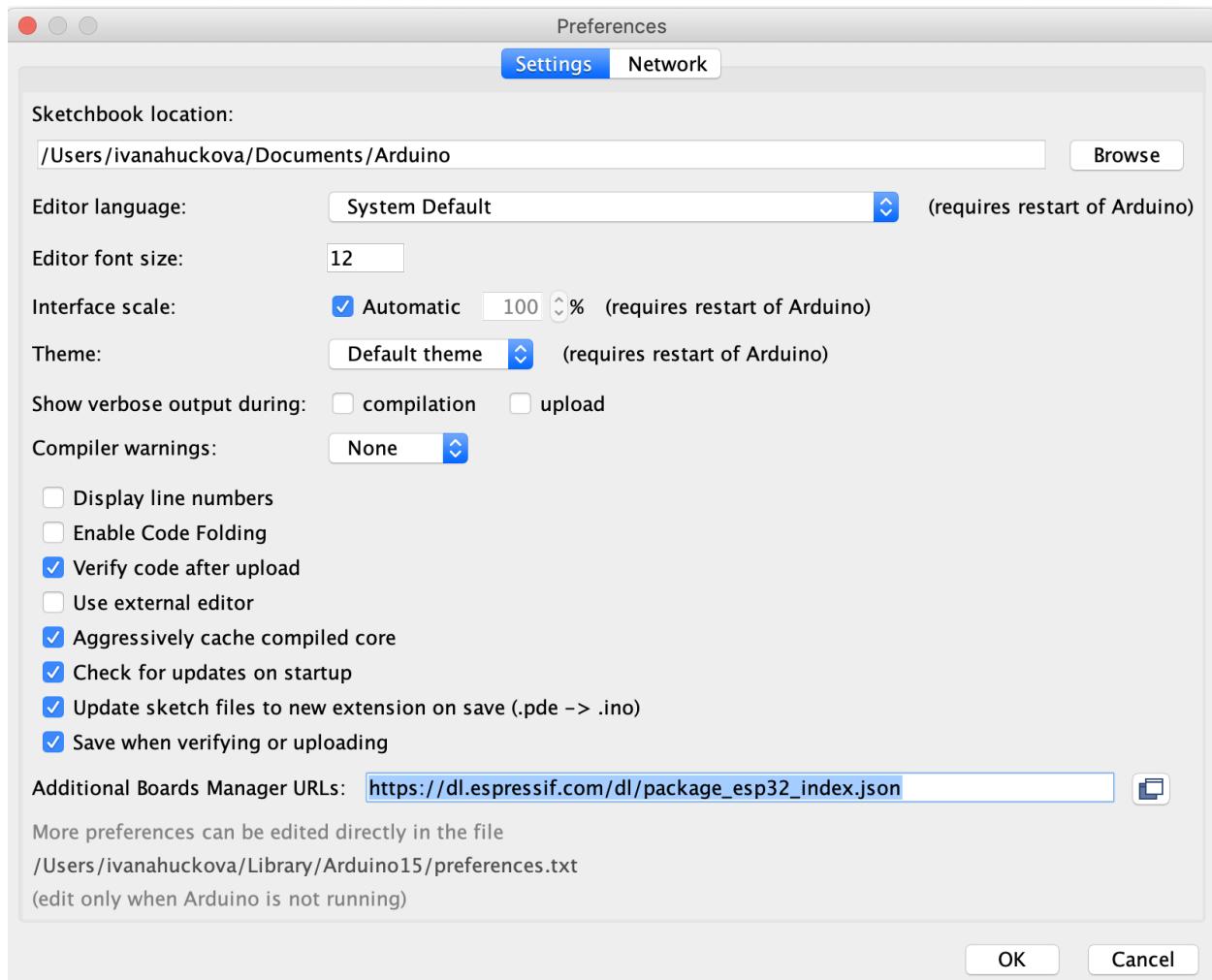


The programming

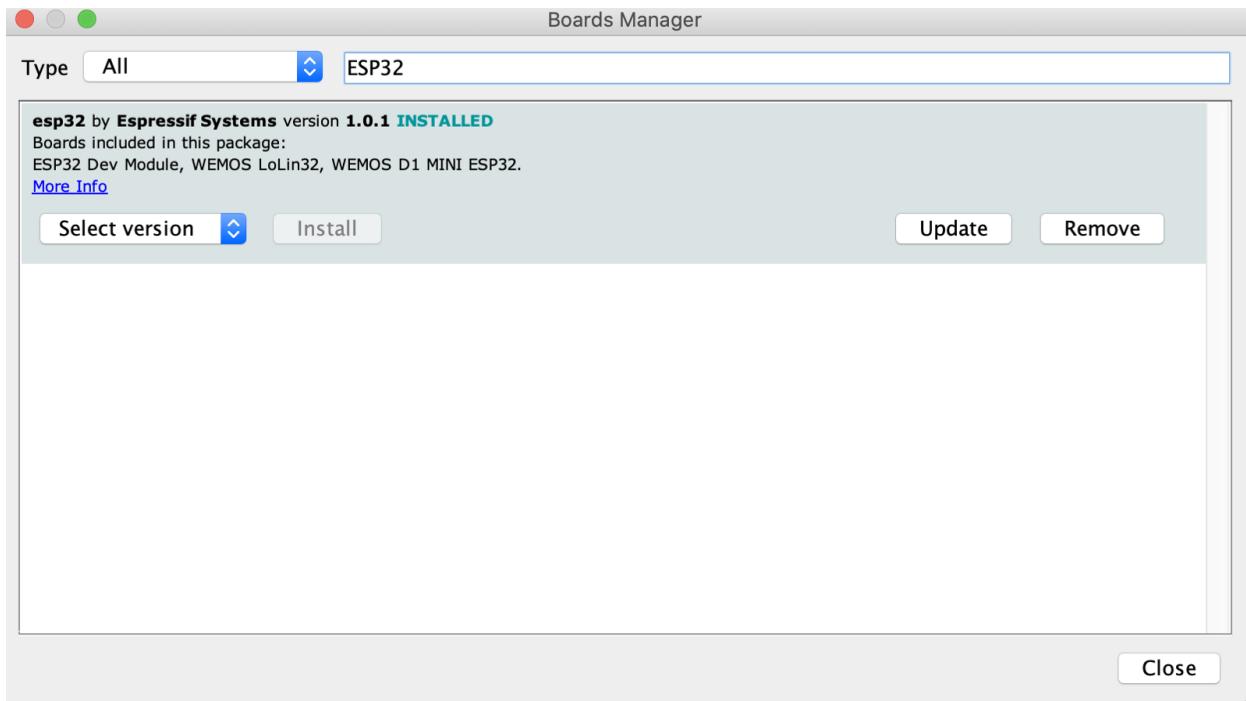
Once I had my sensors and development board ready, I started with writing a program that is going to retrieve data from the sensors and send it to the Graphite and Loki database. I am using an Arduino IDE to upload the program to the development board. Therefore, as a first step, I have connected my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



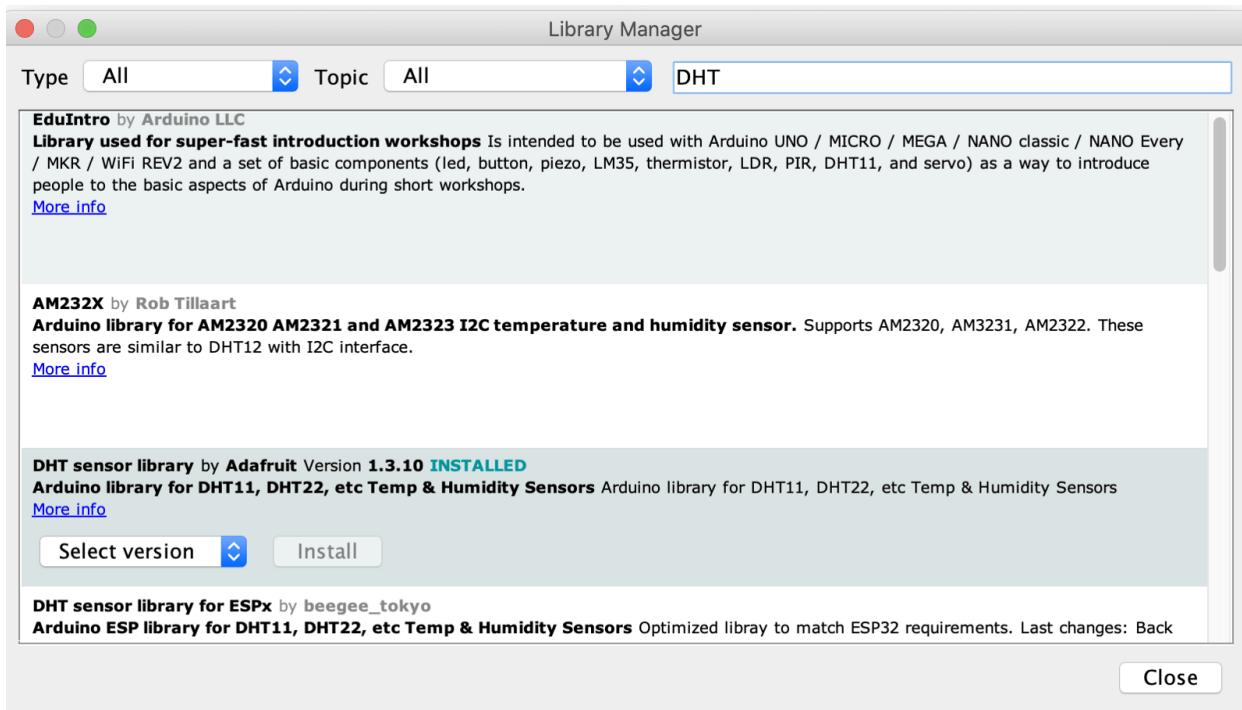
Lastly, in Tools > Boards, I have chosen **DOIT ESP32 DevKit**. This tells the Arduino IDE which profile and base libraries to use when compiling the firmware image, and how to flash it to the board. Make sure that in the Tools > Port submenu, you've selected the new COM port.

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.

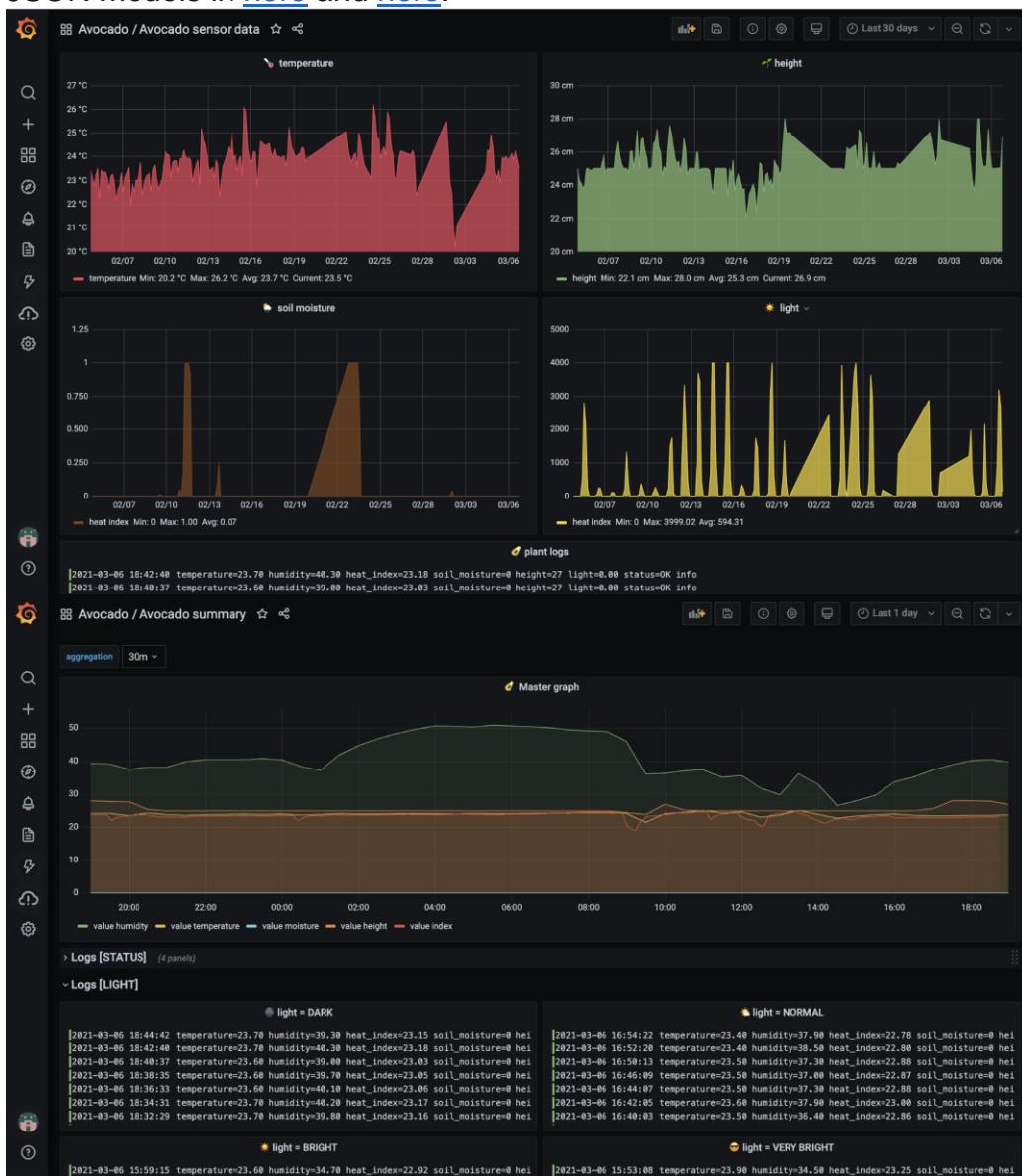
Step 4: Uploading and running the program to ESP32

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.



<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.

[Get Started](#)

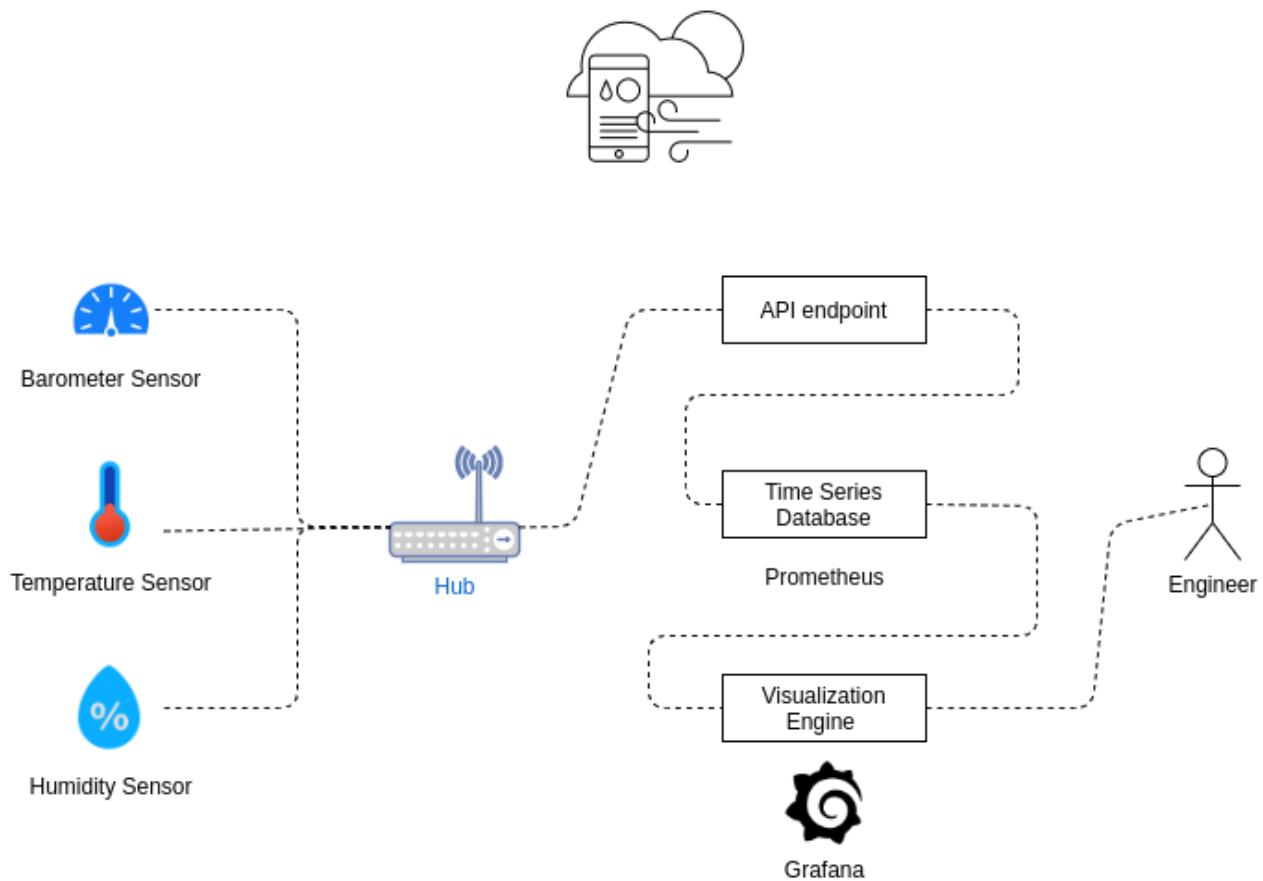
14-day trial

No Credit Card Required

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.

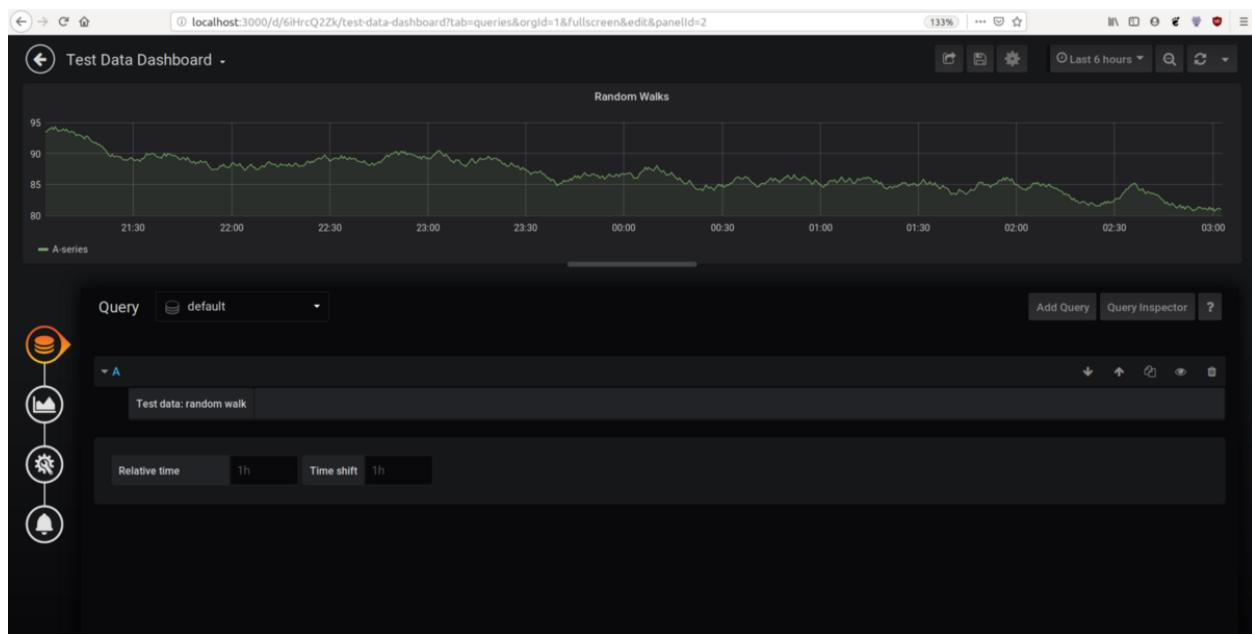
To get these metrics into Prometheus, you will need to set up your Prometheus to scrape the API endpoint visualized in the above diagram. To see one method for doing this, check out this tutorial [here](#).

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

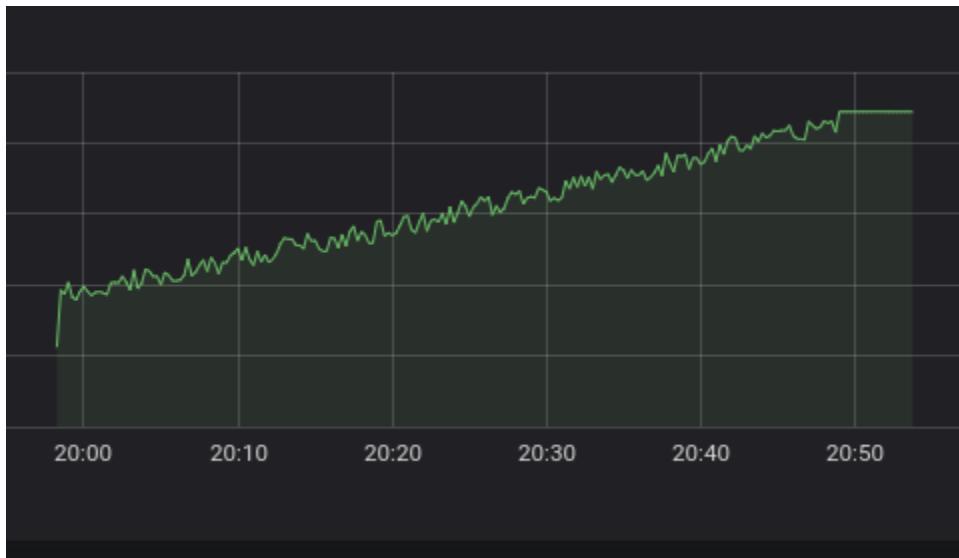
To start building your dashboard, go to the plus sign on the left hand side of the home page, and add a new dashboard. Then add a new panel and add a query. You can see the beginning of a new dashboard in the image below:



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':



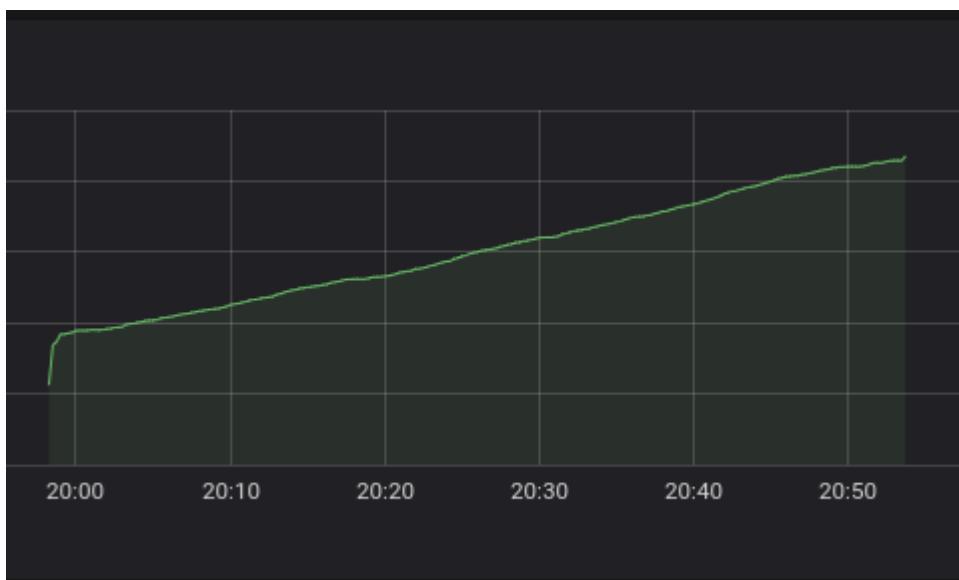
If you want to remove the noise, we can apply the following PromQL function when we query the metric. Just write the following query into your query expression field:

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

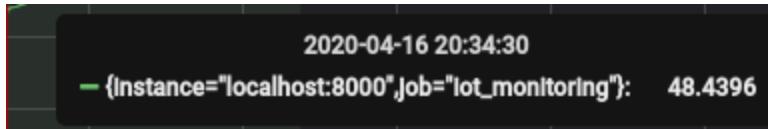
This PromQL expression transforms the query in two ways.

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:

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:

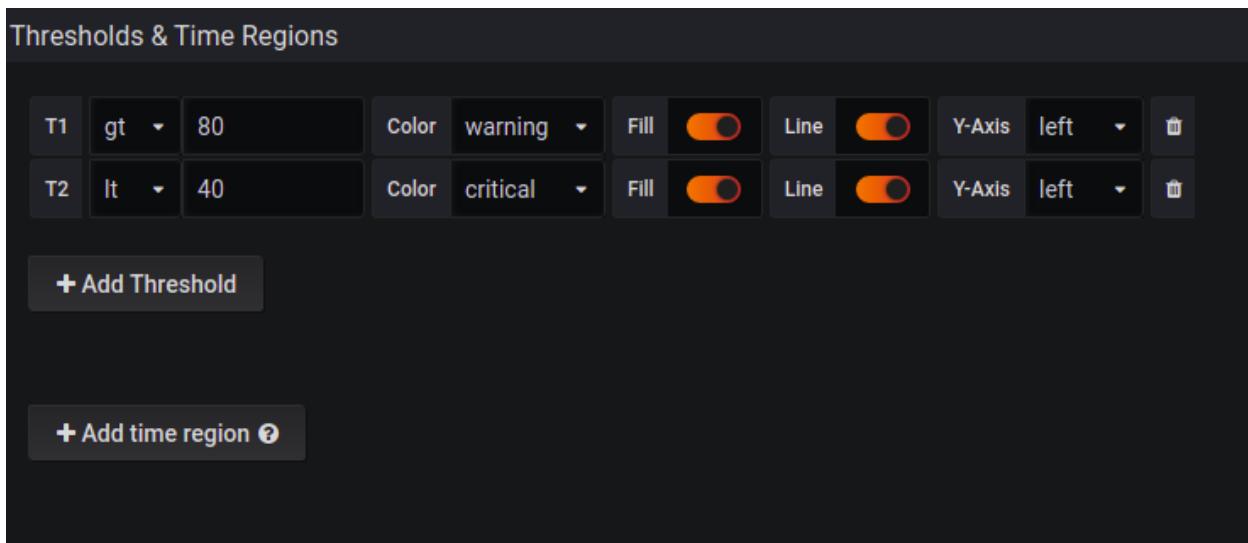
```
bottomk (2, temperature_gauge_c)
```

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:



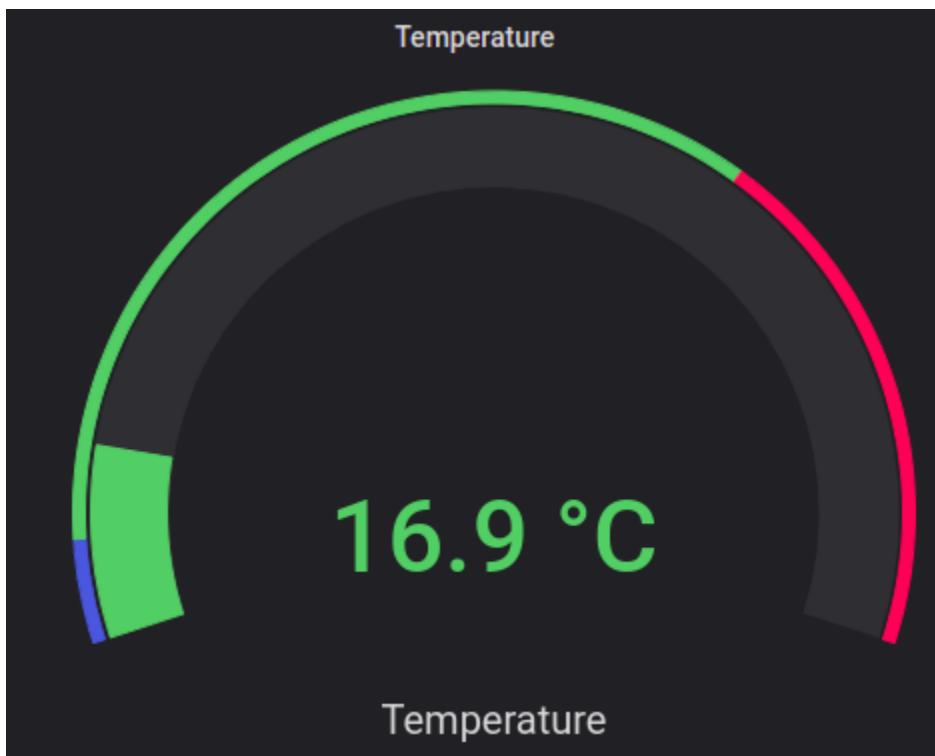
You can set thresholds, where the colour on the gauge will change if the value goes over a certain limit. It’s also great to have configured [alert rules](#) to react as fast as possible. We can use the following query to detect if the instance is healthy:

```
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

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.

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

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

จ้าหัวตอนที่ 1 แล้ว แสดงว่าต้องมีมากกว่า 1 ตอนแน่ แต่ไม่ต้องห่วงว่าจะนาน มีคนปาร์มาไว้ไว้ให้เวลาตั้ง 6 เดือนแทน และคงทำไม่ได้ป้าคดคด ไหนๆก็ให้นาแระ เพียงจะเลยรื่อง Mesh Network ซึ่งก็ไม่ใช่รื่องใหม่อะไร ซึ่งรื่อง Mesh Network ที่จะพูดนี้ก็จะแบ่งออกเป็น 4 ตอนด้วยกัน ไหนๆพูดแล้วก็เล่นใหญ่ไปเลยนะอ้อเจ้า บทความนี้เพียงเอาจมันสักวันๆ ก็อ่าวทบทวนความรู้กัน ไม่ครามมา นานๆ ที่สัก ก่อนอ่านควรทำความรู้การใช้งาน Arduino IDE และบอร์ด MCU โดยเฉพาะ ESP8266 และ ESP32 มาบ้าง

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

Introduction

อะมาตรฐานกันจะหน่อยเดี๋ยวก่อนว่า Mesh Network เป็นยังไงก็คือโครงสร้างของ Network แบบหนึ่งแหล่งที่มีลักษณะเป็น Node ซึ่งชื่อมต่อถึงกันหมวดและทำงานร่วมกันในการส่งผ่านข้อมูลจาก node นึงไปยังอีก node นึง ซึ่งการทำงานในการส่งผ่านข้อมูลตรงนี้แหล่งที่ไม่ใช่เครื่องต่อตัวกันแบบเดียวตัว นั่นหมายความว่าโครงสร้างของ mesh นั้นสามารถปรับเปลี่ยนได้ตลอดเวลา คุยกันตลอด เช่นถ้าจะส่งข้อมูลจาก A ไปจุด B จะไปเส้นทางไหน ถ้าเส้นทางจาก A → B → C → D แล้วถ้า Node B ล่ม จะ route ข้อมูลไปที่ Node ไหนต่อเพื่อให้ลิงป้ายทาง (Self-Organize / Self-Configure) ซึ่งมันจะคิดมาๆเลขในการติดตั้งๆ โอนอุปกรณ์เข้าไปใน Mesh และมันก็คุยกันเองได้ เจ็บะหละ

ซึ่ง Concept ของ Mesh Network คร่าวๆก็อย่างที่อธิบายไปข้างต้นในเนตมีให้อ่านเพียบที่ไทยและเทศ ซึ่งที่ตอนนี้เราเห็นใกล้ตัวเรา กันมาก ก็จะเป็นอุปกรณ์ประเภท Home Automation ซึ่งเริ่มเข้ามาทำการตลาดในบ้านเรายอะชั้น ในตลาดก็จะมีมาตรฐานของการสื่อสาร ผ่านอุปกรณ์พวกนี้อยู่ 2 ค่ายใหญ่กันก็คือ Zigbee และ Z-Wave โดยคุณสมบัติของทั้งสองค่ายนี้เลยก็คือ Mesh Network และ Low power Consumption ละนั้นในการติดตั้งอุปกรณ์ไม่ว่าจะ Zigbee หรือ Z-Wave เข้ากับ Gateway นั้น สามารถทำได้เจ้ามาก ไม่ต้อง config routing หรือเดินสายอะไรให้ยุ่งยาก แฉมยังขยายพื้นที่การใช้งานออกໄປไปไกลขึ้นตาม node ปลายทางที่ซึ่งมันถึงกันด้วย

คราวนี้เรามาดูผู้ที่ใช้ micro controller กันบ้างซึ่งในตลาดก็มีหลายค่าย ถ้าใครอยากรู้ว่า mcu ที่มี z-wave ในตัว ใช้ Arduino IDE พัฒนาและรองรับการทำงานแบบ Mesh Network ของค่าย z-wave ได้ก็มี แต่วันนี้เราจะคุยกันถึงพระเอกตัวหลักของเราที่คือ ESP8266 กับ ESP32 จากค่าย Espressif

เจ้า ESP8266 / ESP32 ที่บางคนเรียกว่าเป็น Arduino Killer นี้มาพร้อมคุณสมบัติ WiFi ในตัวซึ่งหมายความว่าที่จะมาทำอุปกรณ์ประเภท IoT โดยที่สามารถทำตัวเองเป็นได้ทั้ง Client/Server ที่มี Built-in WebServer ในตัว เป็น Access Point ได้ สามารถพัฒนาซอฟต์แวร์ที่เป็น WiFi Enabled ซึ่งทาง Espressif ที่เป็นคนพัฒนา ESP8266/ESP32 ที่เลือกให้เนื่องที่ห่วยๆท่านเห็นแหล่งที่มา

ESP8266 uses mesh network as shown in Figure 1-1. As a result, a large number of nodes can connect to the internet without any improvements of the current router.

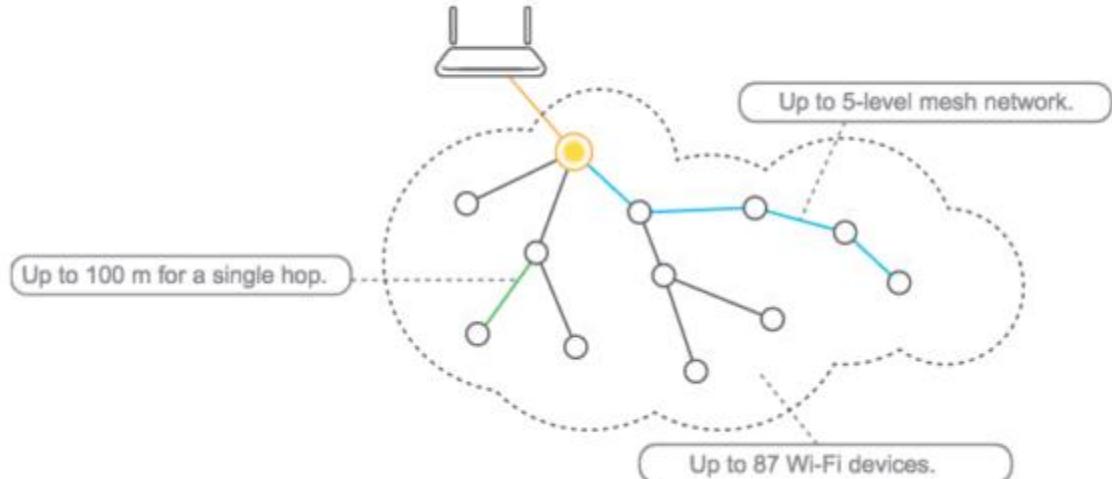
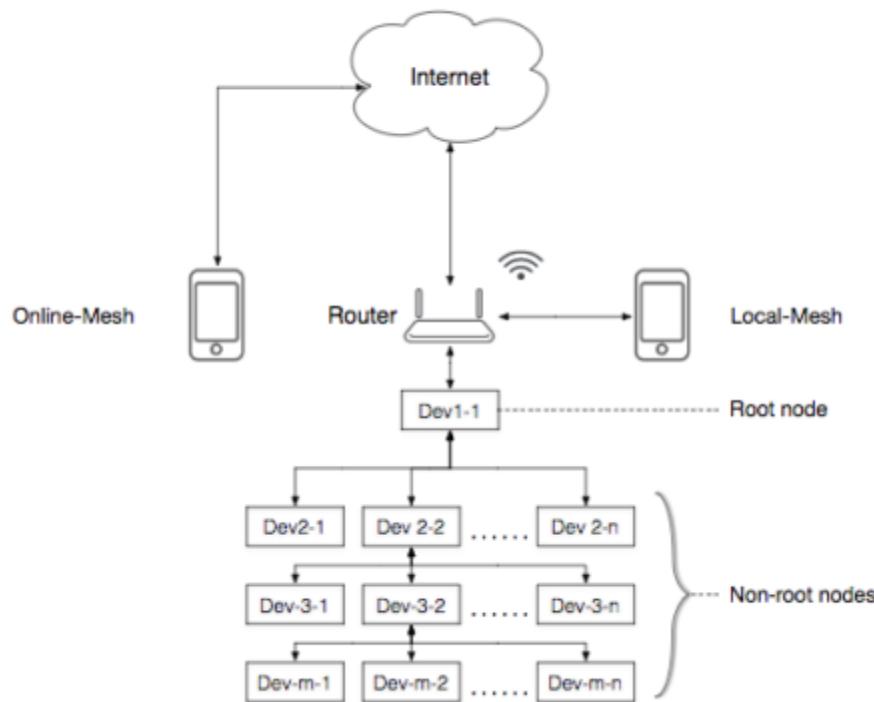


Figure 1-4 shows the mesh network diagram.



ชิ้งที่ดูจากเอกสารของ ESP8266 แล้วเนี่ย สามารถที่จะเชื่อมต่ออุปกรณ์กันได้ถึง 87 ตัวเลขที่เดียวแล้วลองคิดดูถ้าในแต่ละ hop ของ Node อ่า่าว่าแต่ 100 เมตรเลข แค่ 50 เมตรก็พอ อุปกรณ์ 87 ตัวนี่ครอบคลุมพื้นที่ได้เยอะมากๆ พังคูดีอีกแล้ว แต่ถ้าไปคูในส่วนของ การใช้งาน ESP-Mesh บอกได้เลยว่ามีนี่ดี รวมถึงจากข้อมูลที่หามานี่คุณจะบัญชาค่อนข้างละเอียดในการใช้งานโดยเฉพาะเรื่อง Self-Organize/Self-Configure จะนั้นเราจะเข้ามายัง ESP-Mesh Protocol กันไป

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

จากการศึกษาค้นคว้าอย่างหนัก (จริงๆก็ไม่เท่าไหร่หรอ กด Google แล้วก็อ่านไปเรื่อยๆแค่นั้นแหล่ะ ยุคนี้ดีกว่าสมัย 2540 ที่ผมเรียน MCS51 อีก) เลยทำให้พบว่ามีคณขอปัญหาจาก **ESP-Mesh Protocol** จริงๆแล้วใช้งานค่อนข้างยาก อย่าไปเชื่อครับลองเปิด pdf ดู มีมีเนื้อหาที่อ่านมาเรื่อยๆมาพับกับ library ที่ชื่อว่า **EasyMesh** กับ **Painlessmesh** อ่านแล้วรู้สึกว่าชิ้นงานแล้วทั้งสองตัว ตัวนึงง่ายๆอีกตัว ไม่ปากหัว พอดีกันไปเรื่อยๆอีกเลยพบว่า **Painlessmesh** เป็นทางคุณ **BlackEdder** (อันนี้ไม่รู้นามสมมติ หรือชื่อจริงนะ) พัฒนาต่อเนื่องมาจาก **EasyMesh** ซึ่งหยุดการพัฒนาไปแล้ว ที่สำคัญ **Painlessmesh** ใช้ได้ทั้ง **ESP8266** และ **ESP32** อีกด้วย

ว่าแล้วก็มาเริ่มกันเลยดีกว่า เกริ่น **Intro** จะเบื้องต้น ไปกันกระป่องของเด่นที่พอเหลือๆไม่ได้ต่อ กับ เช่นชื่อรุ่นๆ หรือทดลองงานอื่นๆ 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**)

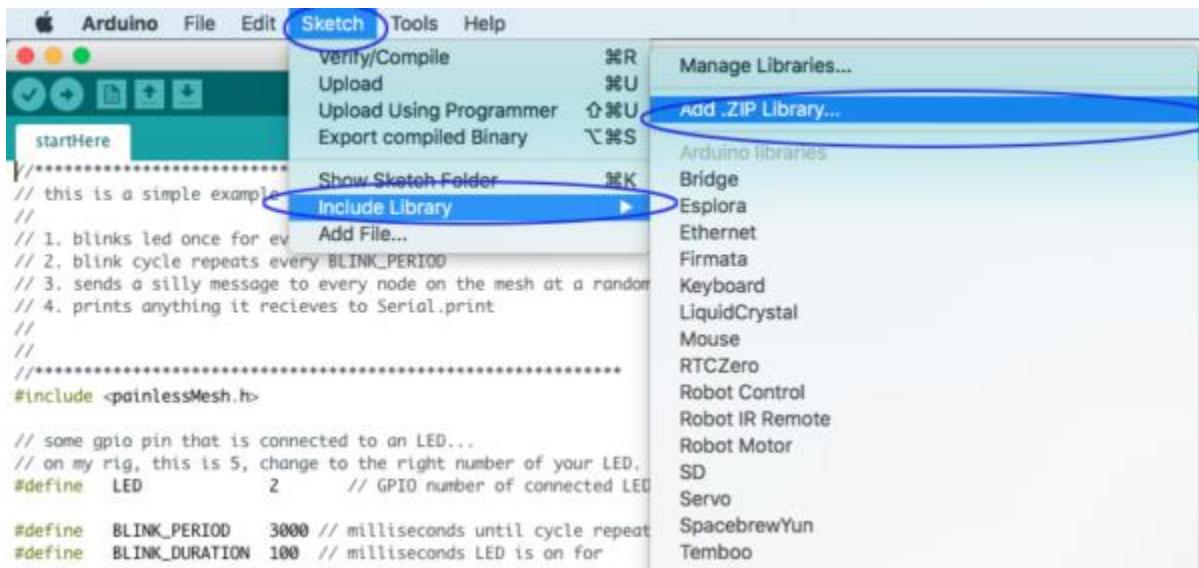
เตรียมความพร้อม

ซึ่งอันนี้ส่วนมติว่าผู้อ่านได้ติดตั้ง **board esp32** และ **esp8266** ไว้เรียบร้อยแล้วนะครับ ขั้นแรกก็ **Download/Clone Library** จาก **Github** มาไว้ที่เครื่องก่อน โดยเข้าไปที่ [Painlessmesh](https://github.com/Coopdis/easyMesh) ที่ **Github**

<https://github.com/Coopdis/easyMesh>

Branch: master	New pull request	Find file	Clone or download
65 commits	1 branch	0 releases	1 contributor
Coopdis committed on Sep 22, 2016 Fixed printf statement in startHere receivedCallback()	1 Latest commit 9d76db7 on Sep 22, 2016		
examples	Fixed printf statement in startHere receivedCallback()	2 years ago	
src	Fixed printf statement in startHere receivedCallback()	2 years ago	
.DS_Store	Fixed printf statement in startHere receivedCallback()	2 years ago	
LICENSE.md	Created LICENCE	2 years ago	
README.md	Update README.md	2 years ago	
library.json	Modifications & additions for publishing as library	2 years ago	
library.properties	Modifications & additions for publishing as library	2 years ago	

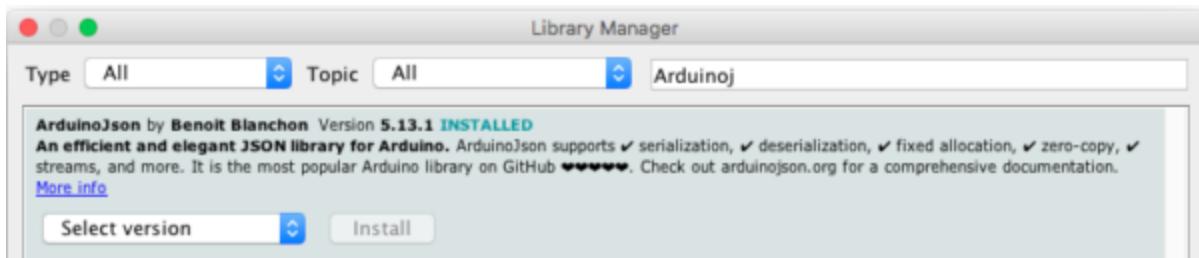
จากนั้นก็ติดตั้งลงใน **Arduino IDE** โดยเข้าไปที่ **Sketch** → **Include Library** → **Add .zip library** แล้วเลือก zip file ที่เรา Download มา



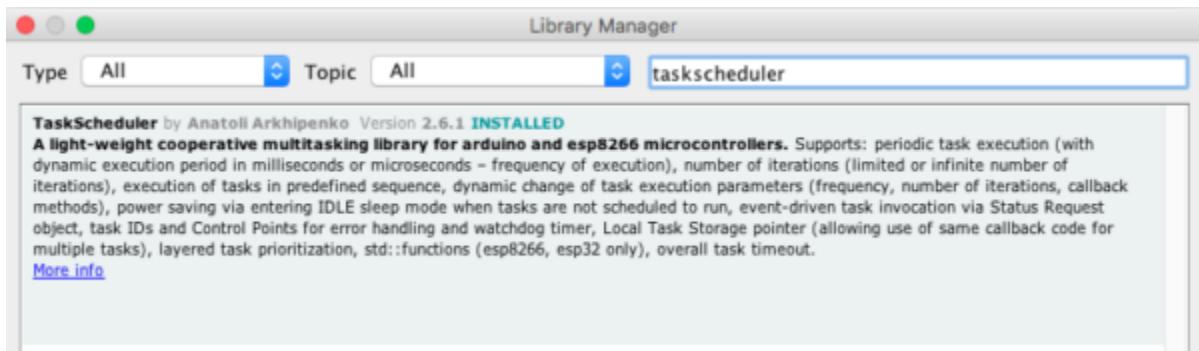
ติดตั้ง Library Dependencies ชื่งจำเป็นต่อการใช้ Painlessmesh

ก่อนการใช้งานเรายังต้องใช้ Library อีกสองตัวนั่นก็คือ ArduinoJson และ TaskScheduler ซึ่งการ Install Library ที่สามารถทำได้ไม่ยากโดยสำหรับสองตัวนี้ เข้าไปที่ Sketch → Include Library → Manage Libraries จากนั้นในช่อง Filter Your Search ก็สามารถใส่ชื่อของ Library ที่สองตัวนี้ได้เลย

ArduinoJson



TaskScheduler



สิ่งที่ต้องรู้และข้อจำกัด

ทางผู้พัฒนา Painlessmesh เดี๋ยวนี้มาอ้างว่า library ชุดนี้เป็น True Adhoc Networking นั่นก็คือไม่ต้องมีการออกแบบว่า node ไหนจะเชื่อมต่อ node ไหน โครงสร้างของ mesh จะเป็นยังไง ไม่ต้องมี router ศูนย์กลาง ซึ่งแค่ 2 node ก็สามารถทำงานได้แล้ว ทำให้คนพัฒนาไม่ต้องมาบุ่นบุนวาญในเรื่องของการจัดการกับ Self-Organize / Self-Configure ไปสนใจในส่วนของ Business Logic / Operation Logic ว่าจะใช้งาน Mesh นี้อย่างไรก็ได้

ซึ่งคำถามของคนใช้ที่นักพัฒนาอย่างเรา เรียกได้ว่าเป็น FAQ ของ Painlessmess เลยก็ว่าได่นั่นก็คือ จำนวน Maximum Node ที่รองรับ โดยที่ทางทีมผู้พัฒนาทิ้งอกไว้ว่า

“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 ขีนอยู่กับ memory ของแต่ละ node นั่นเอง เพราะมันจะต้องใช้ในการเก็บสถานะของแต่ละ node เพื่อที่จะได้ทำ routing config ต่างๆ บนนั้นถ้าแต่ละ Node ย่านค่าจาก Sensor ตามรอบเวลาที่กำหนด แล้วส่งผ่าน Mesh Network ไปเก็บไว้ที่ Server จำนวน memory ที่ใช้ก็ไม่น่ามาก ซึ่งเดียวเราจะมาดูกันว่า memory นี้จะถูกใช้ไปเท่าไหร่เมื่อเพิ่มนode เข้าไปใน Mesh Network

- non-ip networking: ในระบบ Network เราจะคุ้นชินกับระบบ IP หรือบางน้อยที่ mac address แต่ว่า Painlessmesh นั้นไม่ใช้ tcp/ip ใน การสื่อสารและไม่ได้อ้างอิงโดยใช้ ip หรือ mac address แต่จะใช้ chipid ซึ่งก็เป็นหมายเลขเฉพาะในแต่ละชิปของ esp8266/esp32 โดยใน library จะถูกเรียกว่า nodeid แทน
- JSON Based: painlessmesh ใช้ระบบการส่งผ่านข้อมูลในรูปแบบของ JSON ถ้าไม่คุ้นก็อาจจะงงๆหน่อย แต่รับรองไม่ยากแน่นอน ซึ่งข้อดีของมันก็คือ Node ที่รับข้อมูลแล้วต้องการส่งต่อไปยัง webserver หรือระบบงานอื่นที่รองรับ json อยู่แล้วก็จะง่ายเลยที่เดียว
- Delay: เริ่มเข้าสู่เรื่องข้อจำกัด ข้อจำกัดแรกเลยก็คือเรื่องของ delay ซึ่งผู้พัฒนา Painlessmesh เดี๋ยวนี้นำให้หลักเลี้ยง เพราภัยในตัว library เองจะมีการวนในครุภัณฑ์ node ต่างๆ เพื่อทดสอบสถานะของ mesh ตลอดเวลา node ไหนอยู่ node ไหนหลุดไป ถ้าเรามีการใช้งาน delay ภายใน loop อาจทำให้ mesh ของไม่เสียหาย ซึ่งทางแก้ไขก็คือกำหนดเวลาให้ใช้ task scheduler แทน
- อัตราการส่งข้อมูล: ทางผู้พัฒนาเดี๋ยวนี้มาอ้างว่าให้คิดคำนวณอัตราการส่งข้อมูลแบบ conservative อะหน่อย ต้องดูให้เหมาะสมกับ application ด้วย เช่นถ้าจะ monitor temp/humid ในคืนแต่ส่งข้อมูลทุก 5 วินาทีก็อาจจะได้ไม่ค่อยสมเหตุสมผลเท่าไหร่ เพราจะทำให้ esp8266/esp32 ที่มีข้อจำกัดเรื่องของการประมวลผลและหน่วยความจำอาจต้องทำงานหนักขึ้น
- Message Dropped: ถ้าแม้ระบบ Mesh Network มันจะเป็นอะไรที่เจ้มากจนคุณเหมือนเป็น fault tolerance / high availability network แต่ก็มีโอกาสทำงานผิดพลาดได้ ฉะนั้น message ที่ส่งผ่านก็อาจหายไปได้บ้างในบางโอกาส ฉะนั้นก็ขึ้นอยู่กับ logic ของ application ที่เราพัฒนาว่าจะ handle ยังไง

ໄอีกข้อกำหนดข้างบนหนึ่ง คือไม่ได้เก่งเที่ยนขึ้นมาเองหรือครับ ทีมพัฒนา Painlessmesh เนี่ยแหลกแนะนำมา ผนว่าก็ยังไงเป็น best practice บ้างก็คือรับ เวลาจะหาข้อผิดพลาดจะได้ไม่ต้องวนหาตั้งแต่สูนย์ จะได้มีเครื่องให้ไว้เช็คที่มันส่งข้อมูลไม่ผ่านเป็นเพราะอะไร memory overflow มี หรือเป็นัญหาจากภาษาที่ไม่join เข้า mesh หรือว่าเราไม่เขียน delay ไว้

Hello Mesh Network

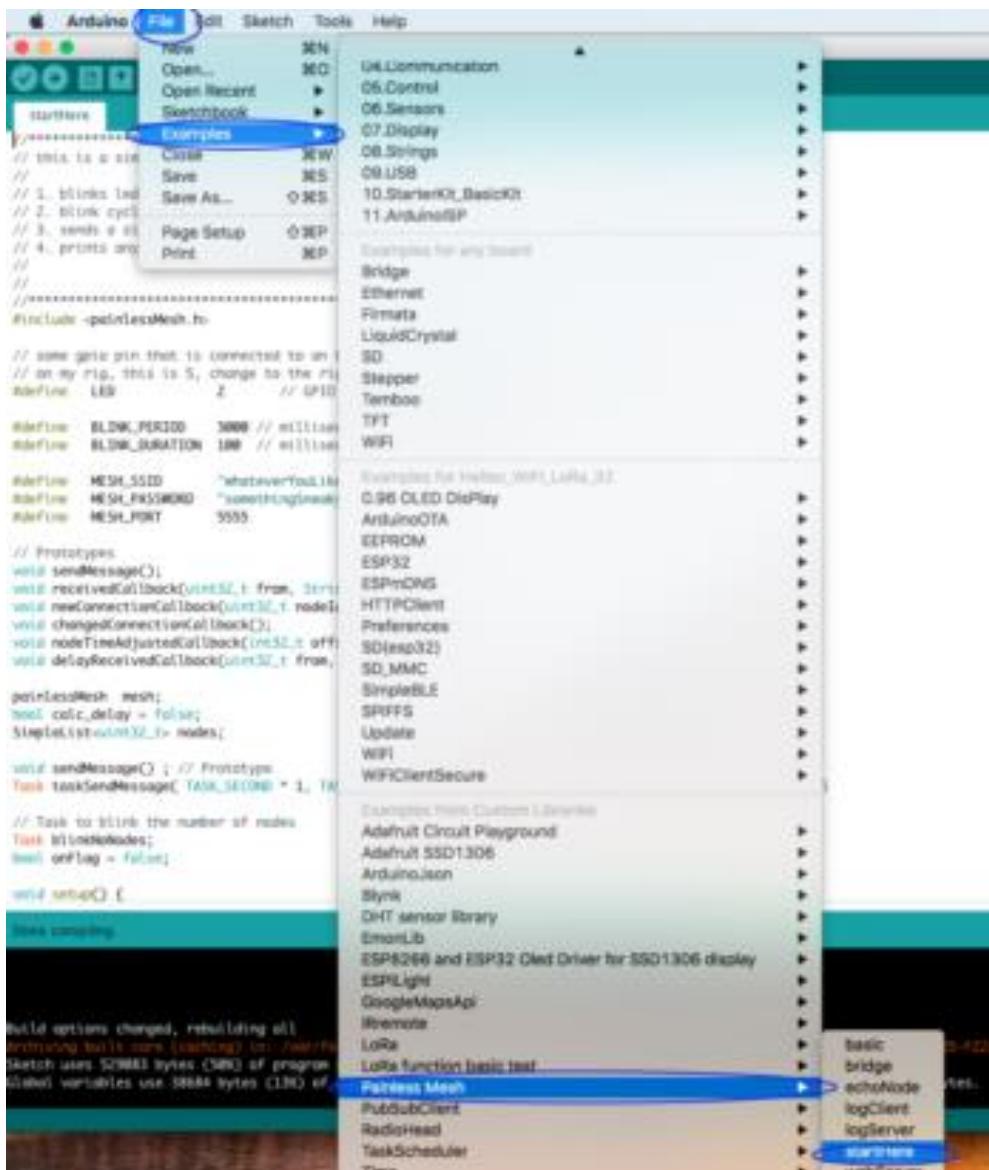
หลังจากเตรียม environment ในการทำงานเรียบร้อยแล้วเราต้องมาเริ่มทดลองสร้าง Mesh Network แรกของเรากันนะอ้อเจ้า เขียนร่ายมาชะยา เริ่มซัก 5 node ก่อนละกัน โดยที่ Node ของเราจะประกอบด้วย



Wemos D1 Mini,

ESP32 Wroom, ESP-01, แล้วก็ Nodemcu อีกสองตัว

ขั้นแรกเลยที่เปิดไฟล์ **startHere** กันก่อนเลยครับ ที่ Arduino IDE ก็เข้าไปที่ **File->Examples->Painless Mesh-> startHere**



มาตรฐานการตั้งค่าร่วมของ Mesh Network เราทัน

```

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

จากไฟล์ startHere ที่มากับตัวอย่าง เราสามารถที่จะแยกงานของ Mesh รวมถึงเพิ่ม Security ในส่วนของ Mesh Network เราได้จาก 3 บรรทัดบนนี้โดยกรับ ชื่อ help ของ painlessmesh รวมถึงคำอธิบายที่อยู่ใน code นั้นก่อนข้างดีเลยที่เดียว ซึ่งการทำงาน Mesh Network โดยใช้ Painlessmesh นั้นจะแบ่งการทำงานเป็นลักษณะของ task โดยการกำหนดจาก task scheduler อย่างเช่น task ที่ใช้ในการส่ง message
`mesh.scheduler.addTask(taskSendMessage);`
`taskSendMessage.enable();`
และในส่วนของ loop ก็จะมีแค่ การ update mesh เป็นหลัก พาก logic ด่างๆจะไปอยู่ที่ task scheduler หมด

```
mesh.update();
```

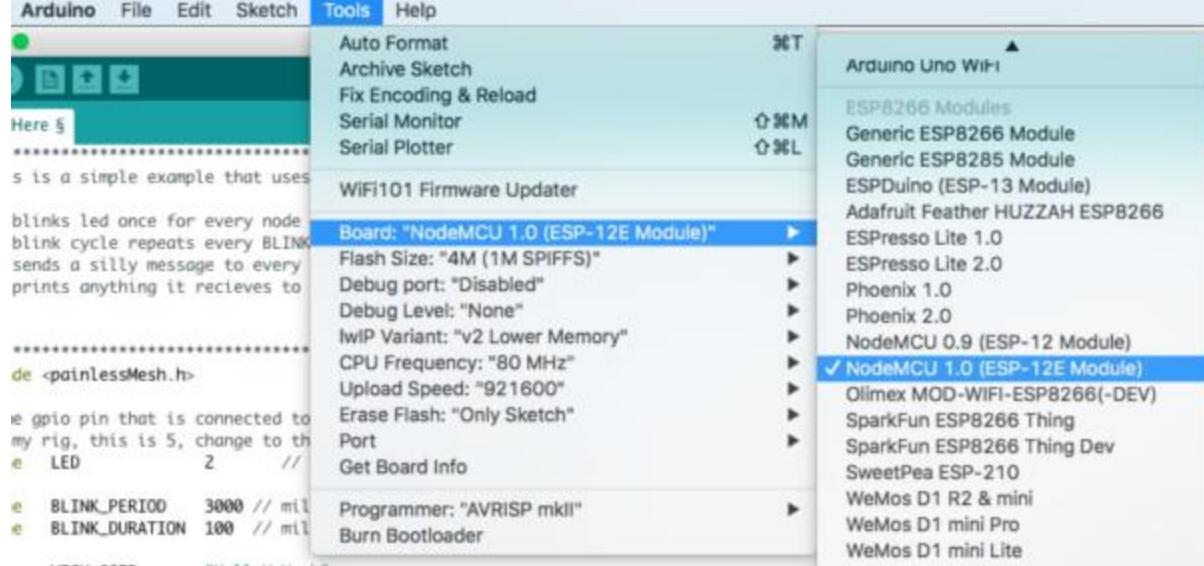
ในส่วนของCallBack นั้นหลักที่ใช้งานจริงๆเลยก็คือ receivedCallback เพื่อที่จะคุ่าว่า message ที่เราได้รับมา ไม่มาจาก播

Broadcast หรือมาแบบระบุ NodeID นั้นเราจะนำไปทำอะไรต่อซึ่งในตัวอย่างนี้ก็คือ Print ออกมาน่าทาง Serial Port

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

}

ซึ่งที่พิมແກ້หลักๆเลยก็คือตัว SSID และ Password แค่นี้จากนี้ก็ทำการ Flash เลขครับ เลือก Board ให้ตรงรุ่นແກ່นี้



ซึ่งคุณจะต้องกดปุ่ม upload ที่อยู่ด้านล่างนี้แล้ว ก็จะสามารถ broadcast ไปยังทุกๆ node ต้องลองทำตามครับตัวอย่างนี้ตัวอย่างเดียวได้อะไร neh

```
// 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 ตัวแรกไปแล้วเปิดที่ Serial Monitor ดู ข้อมูลมันก็จะ酵ะๆหน่อย เพราะมีการ enable debug ไว้ในตอน setup `mesh.setDebugMsgTypes(ERROR | DEBUG | CONNECTION);` ซึ่งระดับของการ debug ก็มีหลายแบบให้คุณเลือกใช้งานตามที่เหมาะสมคุณรับ

```

0x8 stationScan(): HelloMyMesh
0x8 espWifiEventCb(): SYSTEM_EVENT_SCAN_DONE
0x8 scanComplete():--> scan finished @ 11078038 <--
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
Sending message: Hello from node 3896055851 myFreeMemory: 35728
0x8 stationScan(): HelloMyMesh
0x8 espWifiEventCb(): SYSTEM_EVENT_SCAN_DONE
0x8 scanComplete():--> scan finished @ 21155879 <--
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
Sending message: Hello from node 3896055851 myFreeMemory: 35728
Sending message: Hello from node 3896055851 myFreeMemory: 35728

```

ชี้ว่าเมื่อ flash เสร็จและเริ่มทำงาน nodemcu ตัวแรกในวง Mesh Network (มีตัวเดียวจะเรียก Mesh สำમายย) ก็จะเริ่มหา AP ที่ร่อเดียกัน บ้านไก่เรื่องเก็บของ node อื่นๆ ด้วย เพื่อที่จะดึงมาเข้าสู่ Mesh นี้โดยข้อมูลจากการ Debug ก็จะได้ nodeid รวมถึง memory ที่เหลืออยู่ค่ะ

คราวนี้เรามา flash nodemcu อีกสองตัวเข้าไปดูบ้างว่าผลลัพธ์ที่ได้เป็นอย่างไร (note ไว้ก่อนว่า nodeid ของตัวแรกคือ 5851 เอาแค่ 4 ตัวสุดท้ายจะได้ไม่ต้องจำเยอะ และ memory ที่เหลือคือ 35728 byte)

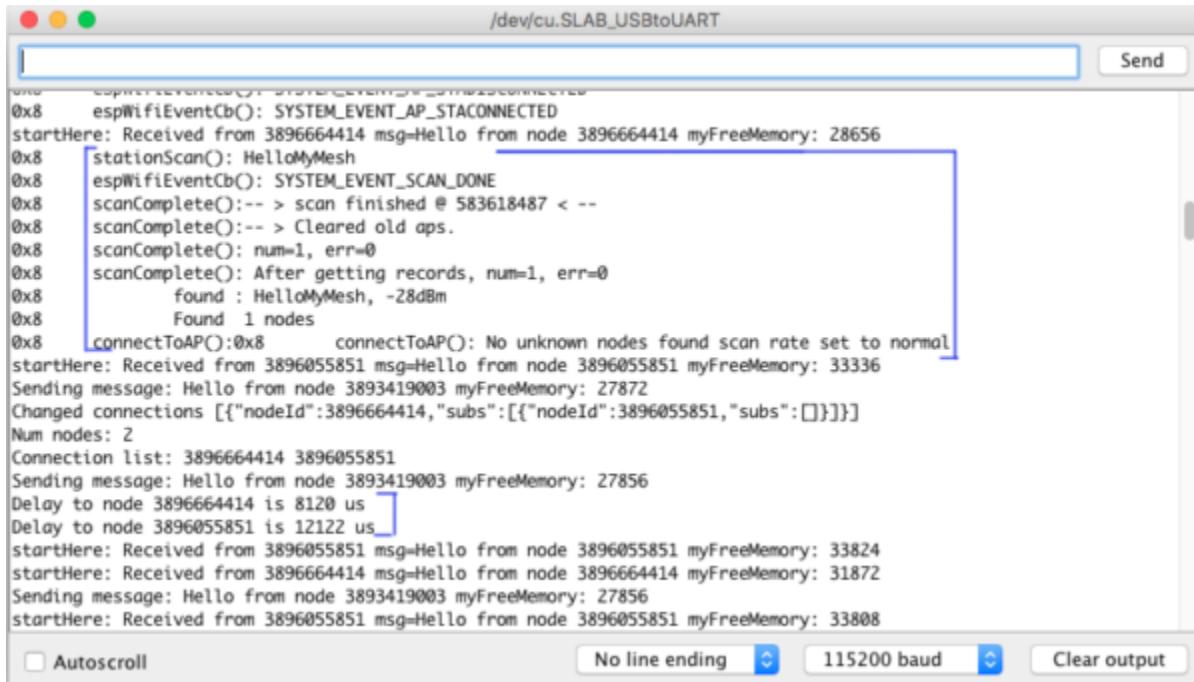
```

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

```

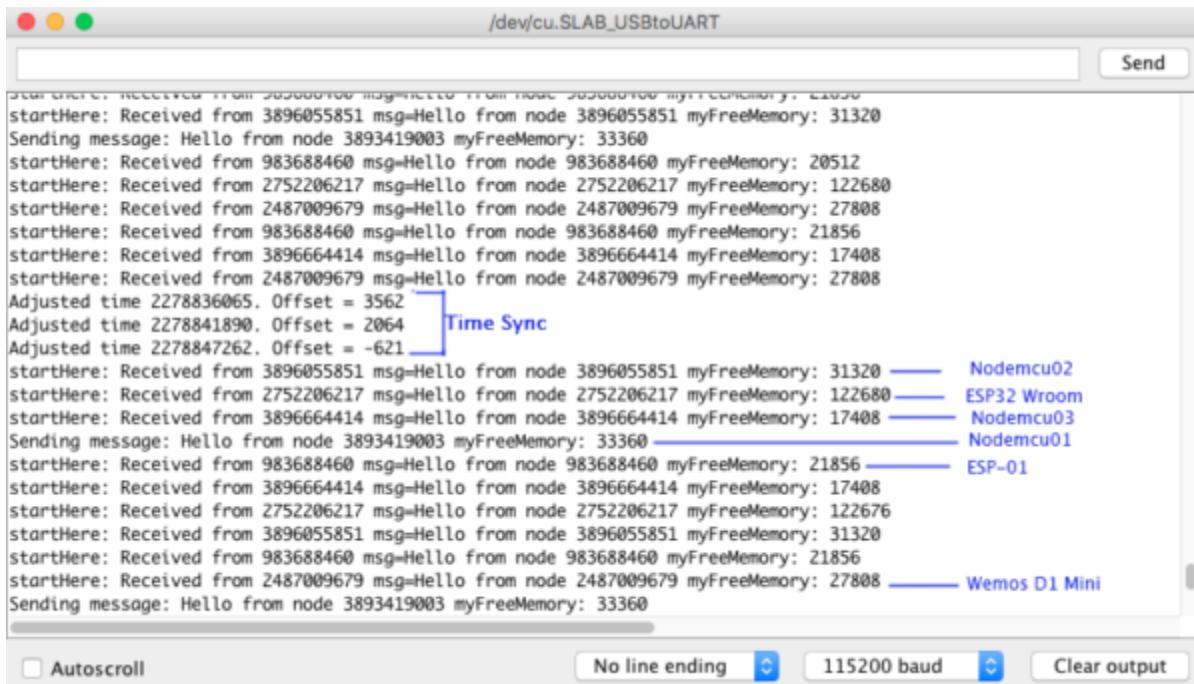
คราวนี้เรามี Node ที่อยู่ใน Mesh นี้ 3 ตัวละนั่นคือ 5851, 4414 และ 9003 ซึ่งแต่ละตัวก็จะ Broadcast Message ออกมากตามเวลาที่สุ่มได้ในแต่ละตัว แต่ในขั้นตอนการกระพริบไฟ Led นั้นจะมีการตั้ง time synchronize เมื่อมี node ใหม่เพิ่มเข้า

ในการ mesh ซึ่งจะมีCallBack ไว้รับและทำงานร่วมกับ MeshUpdate ที่วน loop หากมี Node ไหน drop connection ไปหรือเมื่อ node ไหน join เข้ามาและมีค่าความแรงของสัญญาณต่ำไปกว่า เพื่อใช้ในการจัดการสิ่งที่ต้องการส่งข้อมูล



```
/dev/cu.SLAB_USBtoUART
[REDACTED] 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 ops.
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: 33336
Sending message: Hello from node 3893419003 myFreeMemory: 27872
Changed connections [{"nodeId":3896664414,"subs":[{"nodeId":3896055851,"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
```

แล้ว 3 Node ยังคงอยู่ใน เรามาเพิ่ม Node เข้าไปใน Mesh Network เราก็อีกดึงค่า ถ้าจากตอนมี 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 ตัดขึ้น (Built-in LED pin ของ ESP-01 อยู่ที่ Pin 1)



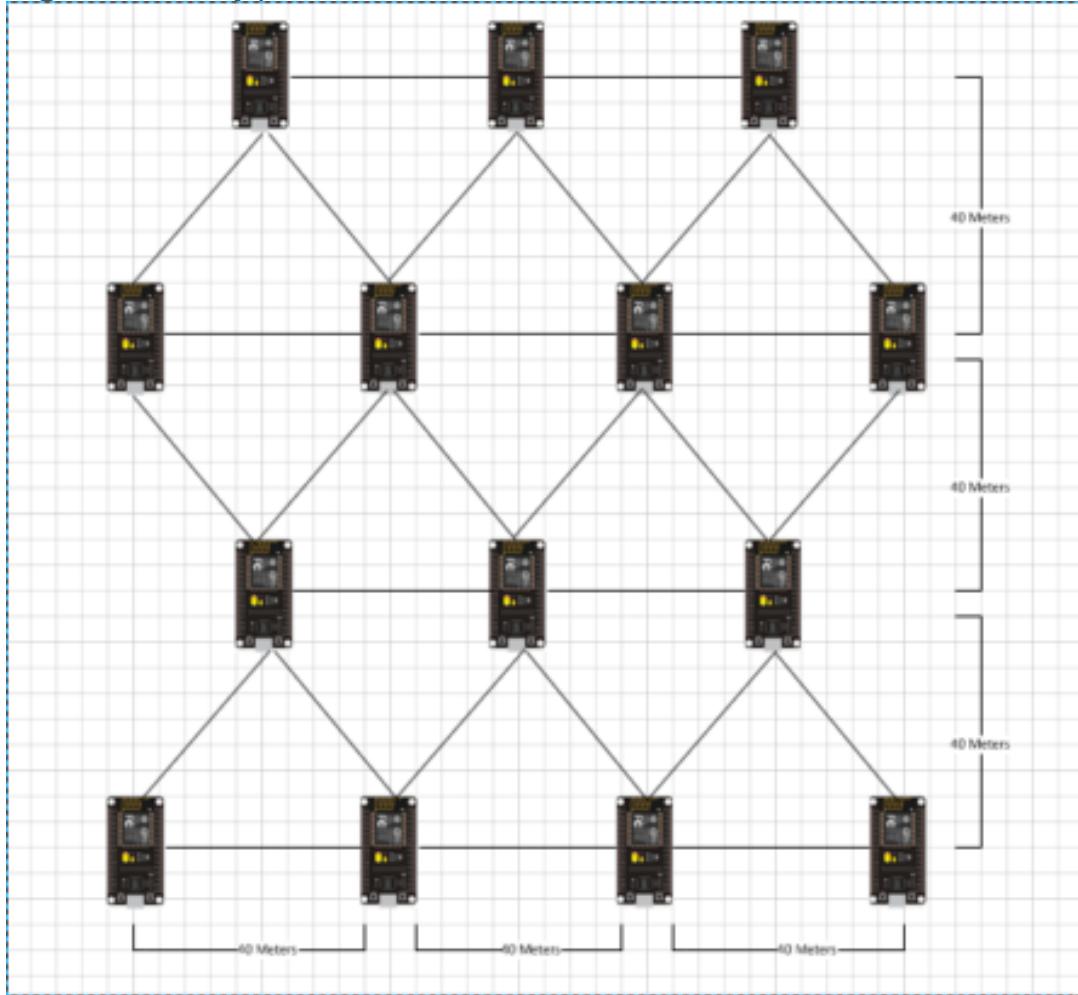
```
/dev/cu.SLAB_USBtoUART
[REDACTED] Send
startHere: Received from 3896055851 msg=Hello from node 3896055851 myFreeMemory: 31320
Sending message: Hello from node 3893419003 myFreeMemory: 33360
startHere: Received from 983688460 msg=Hello from node 983688460 myFreeMemory: 20512
startHere: Received from 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
Sending message: Hello from node 3893419003 myFreeMemory: 33360 Nodemcu01
startHere: Received from 983688460 msg=Hello from node 983688460 myFreeMemory: 21856 ESP-01
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
```

จากรูปด้านบนเมื่อทำงานไปได้ซักพักถ้าคุณ Memory ที่เหลืออยู่ต่ำสุด จะเป็นของ Nodemcu03 ซึ่งเหลืออยู่ 17408 Byte ส่วนที่เหลือมากสุดก็เดาไม่ยากเลย ย่อมเป็น ESP32 MCU ตัวใหม่แน่นอนอยู่แล้ว ดังที่ทางทีมผู้พัฒนาเค้าแจ้งเอาไว้ว่าตัว mesh network นั้นรองรับได้กับ node น้อยที่ memory เป็นหลักเลย ไว้มีเวลาตอนที่ 2 จะเพิ่มนode เข้ามาใน mesh network พร้อมกับเพิ่มการส่งข้อมูลที่อ่านได้จากเซนเซอร์ต่างๆ แล้วมาดูกันว่า ขนาดที่เพิ่มนี้พร้อมกับ operation logic ที่เพิ่มนี้ จะทำให้โครงสร้าง mesh นี้เปลี่ยนไปยังไงกันบ้าง รับรองว่าตอนต่อไปจะได้นำ mesh network มาใช้งานในการส่งข้อมูลกันจริงๆ ละ ส่วนตอนต่อๆ ไปอีกจะมีอะไรบ้าง ก็ลิستตื้นๆ มาให้คุณเรียก哉กันชะหน่อຍ

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

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 จึงเรามีภาพนัดกันหน่อยละกันนะ

High Availability / Fault Tolerance

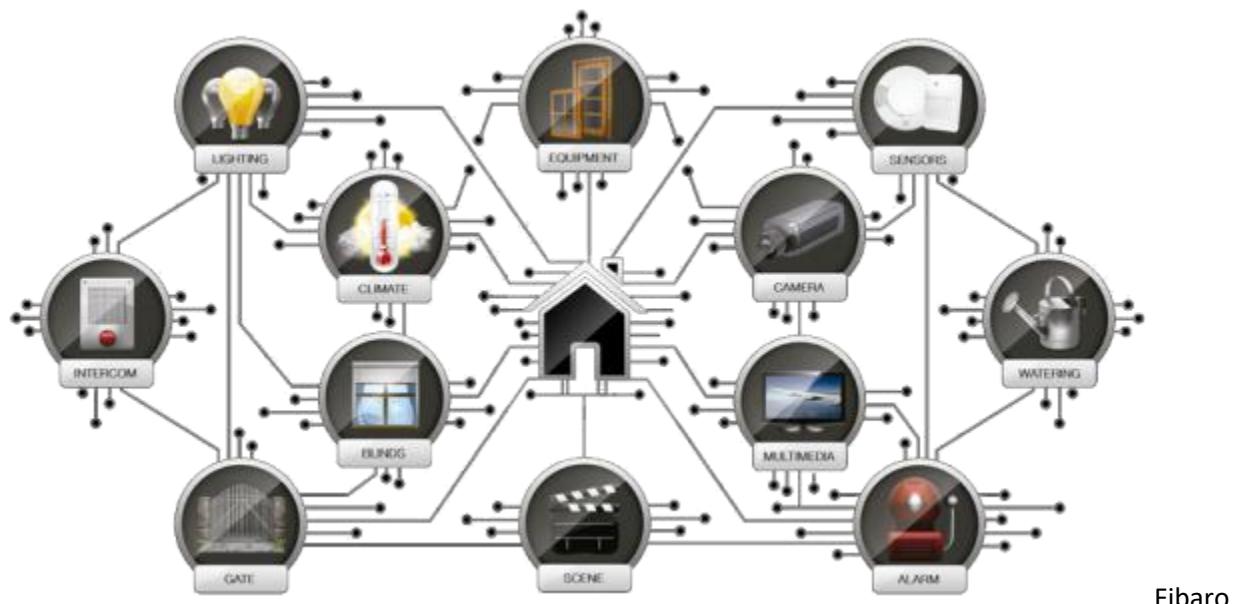


ขยายความจากตอนเดิม Mesh Network นั้นข้อดีก็คือเรื่องของความสเถียร สะดวก ແບນไม่ต้อง setup อะไรมาก ถ้าคุณต้องการเชื่อมต่อ Node กับ Node หนึ่งร่วงไปหรืออาจชำรุดก่อนหนึ่งตัวก็ได้ เส้นทางในการส่งข้อมูลก็ยังคงไม่ถูกตัดขาด เพราะด้วยคุณสมบัติของ Self-Organize / Self-Configure ก็จะทำให้สามารถส่งข้อมูลไปยังปลายทาง หรือ Node ที่ยัง Active อยู่ได้ โดยไม่ต้องมีอุปกรณ์เป็นศูนย์กลางเหมือนระบบ WiFi ที่มีการทำงานแบบ Star ซึ่งต้อง WiFi Router หรือ AP ล่ม ทุก Node ก็จะบ

Coverage Area

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

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



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

ในระบบ Home Automation ที่ใช้ Zigbee หรือ Z-Wave หรือระบบซึ่งรองรับการสื่อสารทั้งสองแบบก็จะมีอุปกรณ์ที่เรียกว่า Gateway ไว้เป็นตัว Bridge ระหว่าง Local Mesh Network กับการสื่อสารผ่านโพรโท콜ที่ต่างกัน เช่น WiFi, Internet หรือแม้กระทั่งข้ามไปมาระหว่าง Zigbee กับ Z-Wave ซึ่งเดียวเราจะต้องที่ 3 เรื่อง Bridge ระหว่าง Mesh Network กับ Network ภายนอกกัน

Power Consumption

ถ้าจุดที่เราติดตั้งมีไฟฟ้าใช้งานตลอดเวลา หรือมีความสูงคงที่ในการเดินสายไฟ ประเด็นเรื่อง **Power Consumption** อาจจะไม่ต้องคิดมากนัก แต่ในระบบ **home automation** มักจะมีอุปกรณ์ประเภทที่ติดตั้งไว้โดยๆ หรือตอนที่บ้านสร้างเสร็จแล้ว อาจไม่สะดวกในเรื่องของการเดินสายไฟ ซึ่งอุปกรณ์เหล่านี้ก็จะเป็นพวก

- PIR ไว้ตรวจความเคลื่อนไหวต่างๆ
- Leak Sensor ไว้คุ้มน้ำร้าวได้ซึ่งคืบล้างงาน หรือได้หลังคาน้ำขึ้น
- IR Remote ซึ่งทำให้เราสามารถตั้งค่าความคุณอุปกรณ์ต่างๆ เช่น TV แอร์ พัดลม

ที่ยกตัวอย่างไป ถ้าบ้านที่สร้างเสร็จไปแล้ว และต้องมาเดินสายไฟ อาจไม่สะดวกนัก **Zigbee** กับ **Z-wave** จึงมาตอบโจทย์ตรงนี้ ในการเชื่อมต่อที่เป็นลักษณะ **MESH Network** ไม่ต้องเดินสาย แฉมกินไฟต่อ ใส่ถ่าน A23 ไปสองก้อนอยู่ได้เป็นปี ทำให้สะดวกมากๆ แต่อุปกรณ์เหล่านี้มีราคาแพง ซึ่งเทียบกับ **ESP8266/ESP32**

ซึ่งถ้าเราจะใช้ **ESP8266/ESP32** แล้วหลักๆ ลิ่งหนึ่งที่ต้องคำนึงถึงเลยก็คือเรื่องของการจ่ายไฟให้ **Node** เหล่านี้ทำงานอยู่ตลอดเวลา ไม่สามารถที่จะ **Sleep** เพื่อที่จะประหยัดไฟได้ เราสามารถดูกันว่าในสถานการณ์ต่างๆ เช่น **nodemcu** ของเรากินกระแสมากน้อยขนาดไหน ซึ่งลองวัดกันแบบหยาดๆ ผ่าน **usb tester** กับ ซึ่งถ้าใครมี **Oscilloscope** หรือ **Multi-meter** อยู่ใกล้ตัวนี่น่าจะวัดได้ละเอียดขึ้น



Scenario1: rom เมล่าฯ วนลูปไปเรื่อย ค่าที่ได้ 80mA

Scenario2: ต่อ WiFi ผึ้งไว้ตามปกติ ค่าที่ได้ออกมา 80mA

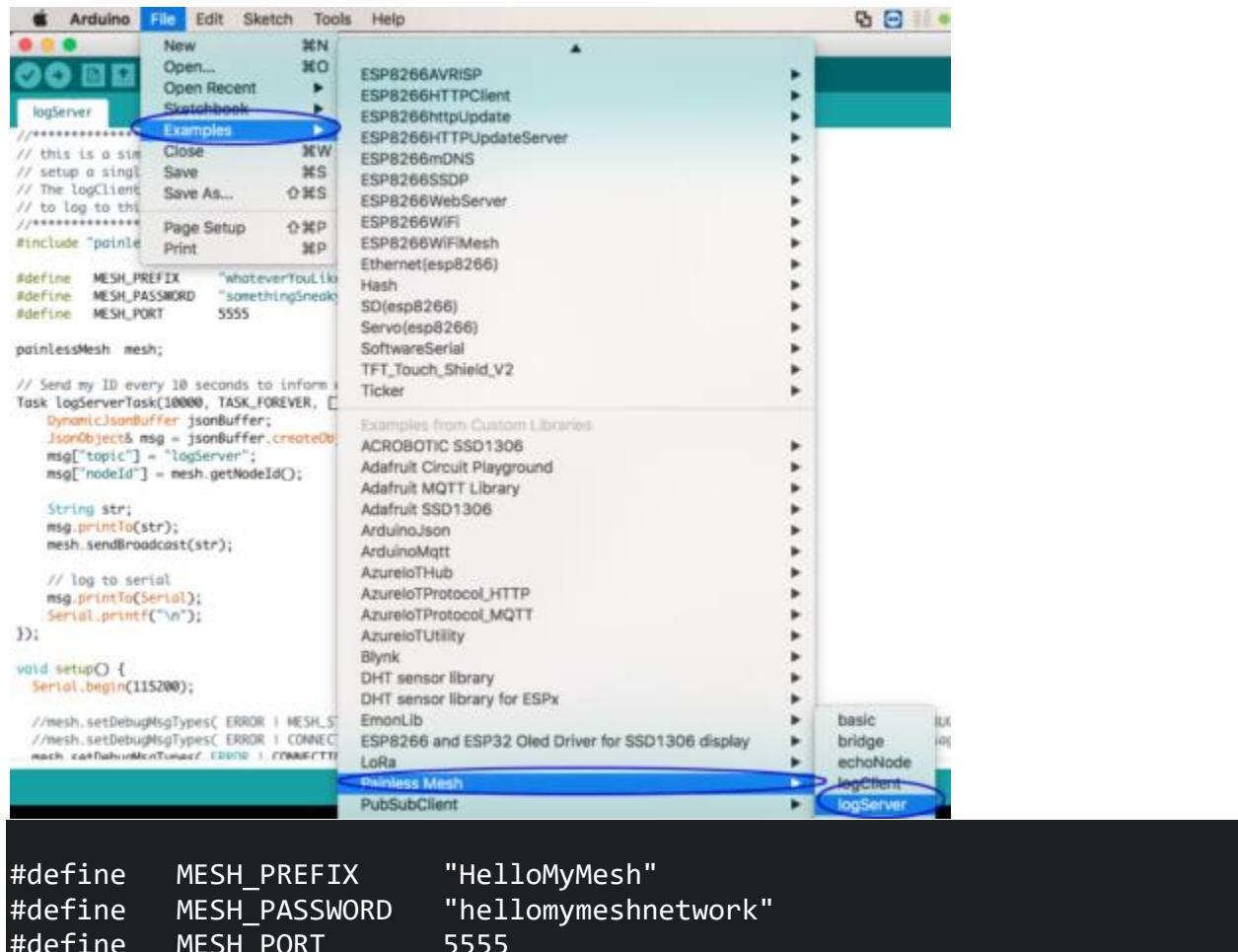
Scenario3: Join Mesh Network (ใช้ Painlessmesh) 2-6 Nodes Broadcast Message ค่าที่สังเกตุได้อยู่ที่ประมาณ 80-90mA มีค่า Peak อยู่ที่ประมาณ 150mA ซึ่งน่าจะมาจากตอน Led กระพริบและตอนที่ Join/Update Mesh Network ซึ่งคาดว่าใช้งานจริงเพิ่ม Node เข้าไป น่าจะทำให้อัตราการกินกระแสสูงขึ้น ถ้าอย่างให้กระแทกตัวลงอาจจะต้องพิจารณาใช้ **ESP12E** อย่างเดียว ตัดพวก LED หรือ Serial to USB IC ออก ซึ่งก็จะทำให้ประหยัดไฟเพิ่มขึ้น แต่ก็ต้อง On Node ไว้อยู่ดีถ้าต้องการให้มีการเชื่อมต่อ

ตอนนี้ถ้าจะเอา ESP8266/ESP32 มาทำ Mesh Network เพื่อรับส่งข้อมูล หรือ Control อุปกรณ์ต่างๆแล้ว น่าจะต้องคำนึงในเรื่องของภาคจ่ายไฟด้วย ในโรงงานหรืออาคารที่ไม่มีปัญหารือการจ่ายไฟ น่าจะเหมาะสม ส่วนงานประเภท Outdoor / Open Field อาจต้องพิจารณาในเรื่องของการนำพา Solar Cell มาใช้ก็เป็นอีกทางเลือกนึงที่น่าสนใจ ซึ่งจากข้อมูลด้านบนก็จะทำให้เราสามารถเตรียมในเรื่องของขนาด Solar Cell และ Battery สำรองไฟไว้ให้เหมาะสมได้ (อย่าลืมวัดจากการใช้งานใน application จะง่าย เพราะ เช่นชอร์บังอย่างกีกินกระแสไฟหมื่นกัม)

ในตอนที่ผ่านมาเราที่ผ่านมาอ่านแล้วได้ทดลองทำตามด้วยตัวเองไป อาจเริ่มตัวอย่างที่สองในส่วนของการทำงานลักษณะที่เป็น Client/Server ไปแล้วก็ได้ เพราะตัวอย่างที่ให้มานั้นค่อนข้างดีเลยที่เดียว แต่ถ้าใครยังไม่ได้เริ่ม ก็มาเริ่มพร้อมๆกันเลยครับ เกร็งจะหายอีกแล้ว

เริ่มใช้งาน Painlessmesh แบบ Client/Server

ตัวอย่างประกอบสำหรับการใช้งานแบบ Client/Server จะใช้อู่ส่องด้วยกันคือ logServer และ logClient โดยสามารถเลือกเปิดได้จาก File—>Examples—>Painless Mesh—> logServer หรือ logClient



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

โดยเราจะเริ่มที่ logServer กันก่อนครับ ซึ่ง concept ก็ยังคงเหมือนเดิม ให้แก้ไขในส่วนของการตั้งค่า Mesh Network ของเรา ซึ่งก็คือ 3 บรรทัดด้านบนนี้ครับ หรือใจจะคิดไว้เหมือนตัวอย่างที่ให้มาก็ได้แล้วก็ Flash ลง ESP8266/ESP32 ของเราได้เลย

แต่ในส่วนของคนที่ใช้ ESP32 ต้องแก้ไขค่านึงครับ เพราะการ Declare ในส่วนของ Authentication Mode นั้นไม่เหมือนกัน แต่ ESP8266 จะแก้ให้เป็นแบบ ESP32 ก็สามารถใช้งานได้เหมือนกันครับ

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

Concept ของตัวอย่าง logServer นั้นคล้ายๆกับตอนที่ 1 ครับเพียงแต่ว่า Broadcast Message ที่ส่งออกไปนั้น มี บุคคลประสงค์ในการส่งออกไปเพื่อบอกให้ node ที่อยู่ใน Mesh Network นี้รู้ว่า พัน NodeID นั้นเป็น Server Node นะ ซึ่งก็ จะทำงานค้าง task scheduler ทุกๆ 10 วินาทีเมื่อมี node ใหม่เข้ามา join ใน Mesh Network ก็จะรู้แล้วว่าจะต้องส่ง ข้อมูลให้ไครซึ่งรูปแบบ Message ที่ส่งออกไปก็คือjsonในฟอร์แมตของ JSON ครับ โดยในส่วนของ receivedCallback ก็จะ เหมือนเดิมคือ ได้รับ message มาที่ Print ออกทาง Serial Port ไม่มีอะไรเปลี่ยนแปลง

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

    String str;
    msg.printTo(str);
    mesh.sendBroadcast(str);           // ส่งข้อความ Broadcast ไปยังทุก Node วาล้วนคือ
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 งานไป เราคือ 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} ข้อความที่ได้รับขึ้นมาอยู่ในรูปของ JSON

```

Autoscroll No line ending 115200 baud Clear output

หลังจาก Flash เสร็จการทำงานก็เป็นอย่างที่เห็นข้างบนครับ จะเริ่มทำการเช็คข้างเดียวว่ามี Node ไหนที่สามารถ Join ได้นำจากนั้นก็ปรับ Offset Time Synchronize ให้เท่ากันเพื่อนำในวงจร แล้วก็เข้าสู่ Task ที่ใช้ในการ Broadcast Message ออกไปว่า เราคือ Server เพื่อให้ Node ทุกด้วยที่อยู่ใน Mesh Network เดียวกันรู้ว่าจะส่งข้อมูลให้ Server ให้ส่งไปที่ NodeID หมายเลขอะไร ซึ่งในที่นี้ก็คือ 6217

มาตรฐานส่วนของ logClient กันบ้าง ซึ่งในส่วนของ receivedCallback เป็นแผลงที่จะเปลี่ยนไป เพราะจะมีการเช็ค Broadcast Message จาก Node อื่นๆ ซึ่งก็คือ Server Node ว่าเทียบยังไงแล้ว Server Node นั้นมี ID นี้นะ จากนี้ถ้าจะส่งข้อความไปที่ ServerNode ให้ส่งไปที่ NodeID นั้นนะ

```

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>()) { // เช็ค Topic ว่าเป็น logServer หรือเปล่า
    // check for on: true or false

    logServerId = root["nodeId"]; // ถ้ามันเท่ากับ 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 ก็จะทำงานต่อ กัน คือถ้าซึ่งไม่เจอกัน ใน Mesh Network ที่เพิ่ง join เข้ามา server หรือเปล่า ผู้นั้นก็ broadcast ข้อมูลของผู้นั้นโดยละเอียดกัน เมื่อจาก Server Node มีระยะเวลา 10 วินาทีถึงจะ Broadcast Message Topic “logserver” ออกไป จนนั้นการทําเช่นนี้ Server Node ที่รู้ข้อมูลจาก Client Node ก็จะได้ข้อมูล แน่นๆ (รวมถึงตัวอื่นๆด้วย)

```

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

    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 ที่ได้รับมา
logServer detected!!!
Handled from 2752206217 msg={"topic":"logServer","nodeId":2752206217} เมื่อเราตกลงกัน Topic Server นี้ก็จะรับบันทึก Serve NodeID
{"topic":"sensor2","value":2} ที่เกิดขึ้นไปในรูปแบบ JSON
logClient: Received from 2752206217 msg={"topic":"logServer","nodeId":2752206217}
logServer detected!!!
Handled from 2752206217 msg={"topic":"logServer","nodeId":2752206217} ที่เกิดขึ้นไปในรูปแบบ JSON
{"topic":"sensor2","value":149} ที่เกิดขึ้นไปในรูปแบบ JSON

```

Autoscroll No line ending 115200 baud Clear output

คราวนี้แน่นที่เราจะจำลองข้อมูลเหมือนในตัวอย่าง ก็ได้เวลาเปิดกระปองและ เอา DHT22 มาพ่วงเข้าไป คราวนี้ Temp/Humid ที่ได้ก็จะ เป็นค่าจริงแล้ว โดยการปรับ Code ในส่วนของ Task ที่ใช้สำหรับการส่งข้อมูล ซึ่งจะมีการอ่านค่าจาก DHT22 และจัดรูปแบบให้อยู่ใน ฟอร์แมตของ JSON โดยที่ Mesh Network จะประกอบไปด้วย 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
    //รีเมงกับ เลข NodeID
    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 ที่ทำการเก็บข้อมูล 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 ครบตัวสุดท้ายที่เปิด Serial Monitor ขึ้นมาจะเห็นได้จากรูปข้างล่างเลยกับว่า เก่อ Node ข้างเคียงอยู่ทั้งหมด 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<---

จำนวน Node ที่เจอทั้งหมด
เลือกเชื่อมต่อเก็บ Node 9679

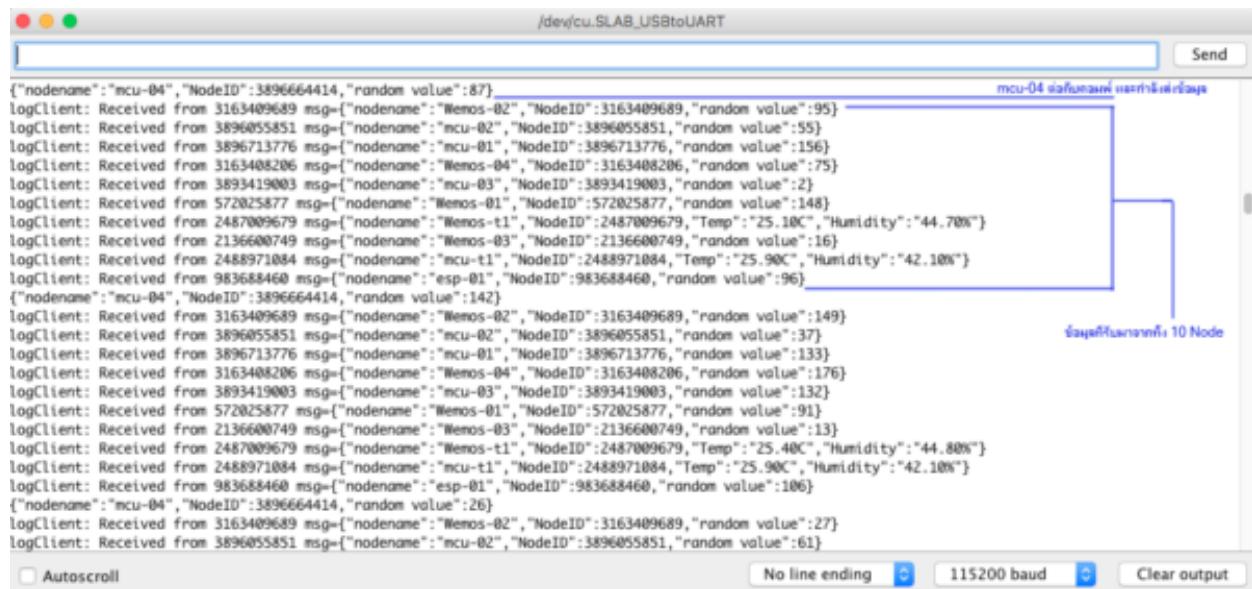
 Autoscroll  No line ending  115200 baud  Clear output

```

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

Scenario ที่ 1: No Server Node

ตามรูปด้านล่างนี้เลยกัน ข้อมูลที่ได้รับผ่าน receivedCallback ได้รับข้อมูลจาก Node ที่ Online ใน Mesh Network นี้ ทั้งหมด เพราะว่ายังไม่ได้เปิด Server



```

/dev/cu.SLAB_USBtoUART
Send
mcu-04 ได้รับข้อมูล แสดงมาให้ดู
[REDACTED]
{
  "nodename": "mcu-04", "NodeID": 3896664414, "random value": 87}
logClient: Received from 3163409689 msg=[{"nodename": "Memos-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": "Memos-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": "Memos-01", "NodeID": 572025877, "random value": 148}]
logClient: Received from 2487009679 msg=[{"nodename": "Memos-t1", "NodeID": 2487009679, "Temp": "25.10C", "Humidity": "44.70%"}]
logClient: Received from 2136600749 msg=[{"nodename": "Memos-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": "Memos-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": "Memos-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": "Memos-01", "NodeID": 572025877, "random value": 91}]
logClient: Received from 2136600749 msg=[{"nodename": "Memos-03", "NodeID": 2136600749, "random value": 13}]
logClient: Received from 2487009679 msg=[{"nodename": "Memos-t1", "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": "Memos-02", "NodeID": 3163409689, "random value": 27}]
logClient: Received from 3896055851 msg=[{"nodename": "mcu-02", "NodeID": 3896055851, "random value": 61}]

```

Autoscroll No line ending 115200 baud Clear output

Scenario ที่ 2: Server Node join Mesh Network

เมื่อเปิด Server Node เข้ามานี้จะเป็นเหมือนรูปข้างล่างนี้ครับ ได้รับ Broadcast Message จาก Server Node และในส่วนของ receivedCallback ก็จะมี logic เพื่อใช้ในการอ่านค่าที่มี Topic ที่เป็น logServer มี ถ้ามีค่าบันทึกเก็บไว้เพื่อในการส่งร่องต่อไปจะไม่ 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 2487009679 msg={"nodename":"Wemos-t1","NodeId":2487009679,"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 แทนดังจะเห็นได้จากปุ่มข้างล่างที่ได้รับ Broadcast Message เข้ามายังเมื่อพิมพ์ข้อความที่มาจากการส่งเมล็ดจาก 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":185}
 {"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

```

คราวนี้เรามาลองดูในฝั่งของ Server กันบ้างว่าเมื่อทุก Node ส่งข้อมูลมาที่ Server Node แล้วข้อมูลที่ได้จะเป็นลักษณะไหนกัน จากรูปจะเห็นได้ว่า ข้อมูลจะได้ครบเหมือนตอนที่เราดูของ Node ใด Node หนึ่งตอนที่ยังไม่มี Server Node เนื่องจากคราวนี้ทุก Node ใน Mesh Network ส่งมาที่จุดเดียว ก็เลยจะได้ข้อมูลครบ และตัว Server Node เองก็ยังคง Broadcast Message ออกไปเรื่อยๆเพื่อให้ทุกคนรู้ว่า พันอยู่ที่นี่นะ ถ้าจะส่งข้อมูลมาหา ให้ส่งมาที่ไหน

```
/dev/cu.SLAB_USBtoUART
Send

logServer: Received from 2487009679 msg={"nodename": "Wemos-t1", "NodeID": 2487009679, "Temp": "23.60C", "Humidity": "38.20%"}  
logServer: Received from 3163408206 msg={"nodename": "Wemos-04", "NodeID": 3163408206, "random value": 139}  
logServer: Received from 3893419003 msg={"nodename": "mcu-03", "NodeID": 3893419003, "random value": 126}  
logServer: Received from 572025877 msg={"nodename": "Wemos-01", "NodeID": 572025877, "random value": 82}  
logServer: Received from 3896664414 msg={"nodename": "mcu-04", "NodeID": 3896664414, "random value": 162}  
{"topic": "logServer", "nodeId": 2752206217}  
logServer: Received from 2488971084 msg={"nodename": "mcu-t1", "NodeID": 2488971084, "Temp": "24.30C", "Humidity": "36.50%"}  
logServer: Received from 2136680749 msg={"nodename": "Wemos-03", "NodeID": 2136680749, "random value": 92}  
logServer: Received from 983688460 msg={"nodename": "esp-01", "NodeID": 983688460, "random value": 154}  
logServer: Received from 3163409689 msg={"nodename": "Wemos-02", "NodeID": 3163409689, "random value": 45}  
logServer: Received from 3896055851 msg={"nodename": "mcu-02", "NodeID": 3896055851, "random value": 100}  
logServer: Received from 3896713776 msg={"nodename": "mcu-01", "NodeID": 3896713776, "random value": 126}  
logServer: Received from 3163408206 msg={"nodename": "Wemos-04", "NodeID": 3163408206, "random value": 20}  
logServer: Received from 3893419003 msg={"nodename": "mcu-03", "NodeID": 3893419003, "random value": 41}  
logServer: Received from 572025877 msg={"nodename": "Wemos-01", "NodeID": 572025877, "random value": 152}  
logServer: Received from 2487009679 msg={"nodename": "Wemos-t1", "NodeID": 2487009679, "Temp": "23.60C", "Humidity": "38.20%"}  
logServer: Received from 3896664414 msg={"nodename": "mcu-04", "NodeID": 3896664414, "random value": 97}  
logServer: Received from 2136680749 msg={"nodename": "Wemos-03", "NodeID": 2136680749, "random value": 67}  
logServer: Received from 2488971084 msg={"nodename": "mcu-t1", "NodeID": 2488971084, "Temp": "24.40C", "Humidity": "36.50%"}  
logServer: Received from 983688460 msg={"nodename": "esp-01", "NodeID": 983688460, "random value": 153}  
logServer: Received from 3163409689 msg={"nodename": "Wemos-02", "NodeID": 3163409689, "random value": 45}  
logServer: Received from 3896055851 msg={"nodename": "mcu-02", "NodeID": 3896055851, "random value": 104}  
logServer: Received from 3896713776 msg={"nodename": "mcu-01", "NodeID": 3896713776, "random value": 33}  
logServer: Received from 3163408206 msg={"nodename": "Wemos-04", "NodeID": 3163408206, "random value": 50}  
logServer: Received from 3893419003 msg={"nodename": "mcu-03", "NodeID": 3893419003, "random value": 142}  
logServer: Received from 572025877 msg={"nodename": "Wemos-01", "NodeID": 572025877, "random value": 179}  
logServer: Received from 3896664414 msg={"nodename": "mcu-04", "NodeID": 3896664414, "random value": 8}  
{"topic": "logServer", "nodeId": 2752206217} Server 01 Broadcast NodeID 0000000000000000  
logServer: Received from 2487009679 msg={"nodename": "Wemos-t1", "NodeID": 2487009679, "Temp": "23.60C", "Humidity": "38.20%"}  
logServer: Received from 2136680749 msg={"nodename": "Wemos-03", "NodeID": 2136680749, "random value": 21}
```

ขยายความในส่วนของ Server Node กันอีกนิดว่าทำไนมต้อง kob Broadcast Server NodeID อยู่ต่อกดเวลา

- ข้อดีเล็กน้อย ถ้าในอนาคตต้องมีการเปลี่ยนอุปกรณ์ในส่วนของ Server Node ที่เสียไป แต่มีอีกชุดอุปกรณ์เปลี่ยน Server Node แล้วอนุว่า ChipID เปลี่ยนไปจะ การ Broadcast Server NodeID และในส่วนของ Client ที่มี receivedCallback ก็จะรู้ว่าต้องส่งไปที่ Server Node ใหม่หมายความอะไร
 - ซึ่งถ้าเข้าใจหลักการที่เขียนมาสองบทนี้ เราใช้งานารถประยุกต์ใช้งานในการทำ Server Node ที่มีมากกว่า 1 คือได้รับการทำระบบ Mesh Network ที่ใช้ในการสื่อสารออกไป Network ภายนอกนั้นเสียช่วงชิ้น หรือจะแบ่งงานภาระของ Server Node เป็นลักษณะของ load balance ก็ซึ่งได้

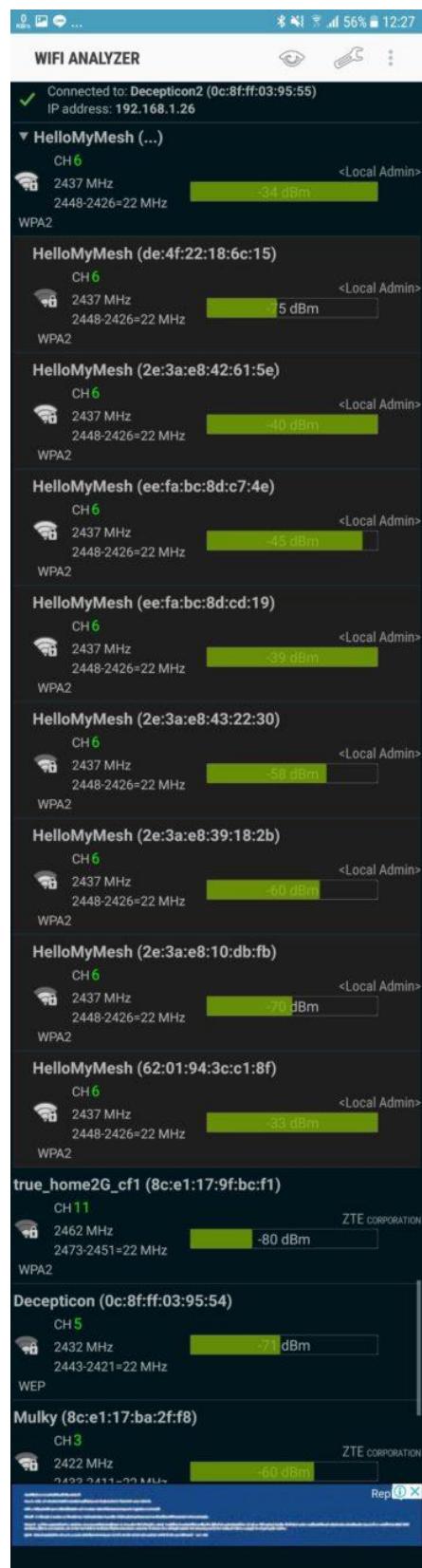
ตัวอย่างการประยุกต์ใช้งาน

- **Read and Forward:** เกสที่จะเพ้าท์สำหรับพากการส่งข้อมูลเพ้า Server อาจจะเป็นการตัวอุณหภูมิ ความชื้น หรือเมื่อกระทึ่งเสียงบริเวณต่างๆของโรงงาน แล้วมาเก็บไว้ที่ Server เพื่อที่จะแสดงผล หรือนำข้อมูลไปวิเคราะห์เหตุต่างๆ เมื่อนั้นในด้านย่าง logServer/logClient ที่เราเพิ่งทำกันไป เป็นการใช้งานที่ค่อนข้าง One-way จะเป็นส่วนใหญ่ และที่เป็น Application ของ iot จะเป็นส่วนใหญ้อีกชั้นกันด้วย
 - เมื่อจากการส่ง message ภายใน Mesh Network นี้สามารถที่จะกำหนดปลายทางผู้รับได้ด้วย NodeID หรือจะส่ง Broadcast แล้วค่อย Filter Message เอก้าได้ว่าปลายทางที่ระบุใน Broadcast Message เป็นของเราหรือเปล่า จากนั้นอ่านค่าที่ได้มา เปรียบเทียบ logic ที่ต้องการ อาจจะเป็นการส่งให้ปิดบิ้มน้ำเพื่อรดน้ำดื่นไม้ หรือสั่นให้ปิด Siren นอกอาการกรณีการตรวจสอบวันไฟໄด จะแจ้งเป็น zone หรือแจ้งเป็นจุดๆ ก็แล้วแต่หลักการเรียกข้อความที่ส่งไป และหลักการเรียกข้อความ logic ของข้อความที่ได้รับมา ซึ่ง Painlessmesh ใช้รูปแบบการส่งแบบ JSON ก็ต้องยังเขียน
 - **Point to Point / Independent:** เมื่อจาก Mesh Network นั้นทำให้ทุก Node นั้นเชื่อมหากันอยู่แล้ว Node ไหน อยากคุยกับ Node ไหนก็ได้ เกสลักษณะนี้ก็อาจจะมี Node ที่ทำงานเป็นลักษณะ Control Panel สามารถที่จะเลือกได้ว่าจะส่งคำสั่งไปที่ node ค้าไหนก็ได้ โดยการอ้างอิงจาก NodeID หรือค่าจะแบ่ง NodeID ให้เป็นช่วงๆ ที่คุณต้องการ เช่น 0-100 หมายความว่า 0-100 นั้นคือช่วงที่ 1 ของ NodeID ที่คุณต้องการสื่อสาร

หมายเหตุ

เนื่องจาก ESP8266/ESP32 ซึ่งใช้ WiFi ย่าน 2.4 GHz อยู่ริบเวก้าใช้งานในโหมดของ Station and AP เพื่อให้ Node ต่างๆมาเกาะรวมกันเป็น Mesh Network แล้ว ถ้าเปิดโปรแกรมพวก WiFi Analyzer มาคุก็จะเห็นชื่อ AP ใน Mesh

ของเรารักษาเพียงเดียว ซึ่งอาจจะไปกวนกับ AP อื่นหรือ AP อื่นอาจจะมีความเร้าใจได้ ยังไงก็ลองพิจารณาการใช้งานให้เหมาะสมดูครับ



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

เรื่องล่าท้ายตอน

สมัยปี 40 เป็นปีแรกเลยที่ได้รู้จัก ได้ใช้งาน micro controller ตระกูล MCS51 โปรเจกแรกๆ ก็หนีไม่พ้นหุ่นเดินตามเส้น ต้องเขียนคำภาษา Assembly หน่วยความจำที่มีจำกัด อุปกรณ์ที่ใช้ค่าทรัพย์ flash ก็แพง อุปกรณ์จะซื้อที่ต้องนั่งรถจากศala มากับบ้านหม้อ เทียบกับสมัยนี้ที่สะดวกสบายขึ้นมาก การเข้าถึง content ด้านต่างๆ ก็ง่ายขึ้น สมัชตอนเรียนสิ่งเดียวที่พอหาอ้างอิงได้คือ โปรเจกจบของรุ่นพี่ แล้วก็หนังสือจากห้องสมุด พอยื้อซู่ทุกที่มี raspberry pi, arduino และ social media ช่างซื้อสิ่งใดก็เปลี่ยนไปเรื่อยๆ เหลือเกิน เดี๋ยวนี้ ใหม่ๆ นี่เรียนรู้กันได้เร็วและ多了มาก ด้วยอาชญากรรมที่มากขึ้น เป็นส่วนงานจากลงมือ implement เอง มาเป็นคุณภาพรวมบริหารงาน ก็งั้นสนุกกับการทำความรู้ใหม่ๆ ทดลองนั่นนุ่นนั่น โอกาสอำนวย ยิ่งเห็นการเปิดตัวของ maker space และ startup ต่างๆ แล้วยิ่งคิดถึงสมัยอยู่ lab ของมหาลัยศึกษาดูน้ำ ในตอนที่ 2 ก็ฝากกันไว้ท่านนี้ก่อนครับ แล้วเดี๋ยวมาต่อตอนที่ 3 กัน

ปล.

ถ้าหาก Sponsor หลัก (และ Sponsor เดียว) ของทีมผู้พัฒนา Painlessmesh แล้ว ก็จะมีบริษัท [Sowillo](#) ที่เป็นผู้ให้เงินทุนสนับสนุนโดยธุรกิจของ Sowillo ก็จะทำให้กับเครื่องทำน้ำร้อนโดยใช้พลังงานแสงอาทิตย์ ซึ่งทำให้เรา Painlessmesh ไปใช้งานในเชิงพาณิชย์ ถ้าสามารถสนับสนุนทีมพัฒนานี้ได้ ก็จะดีมากๆ เลยกัน

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

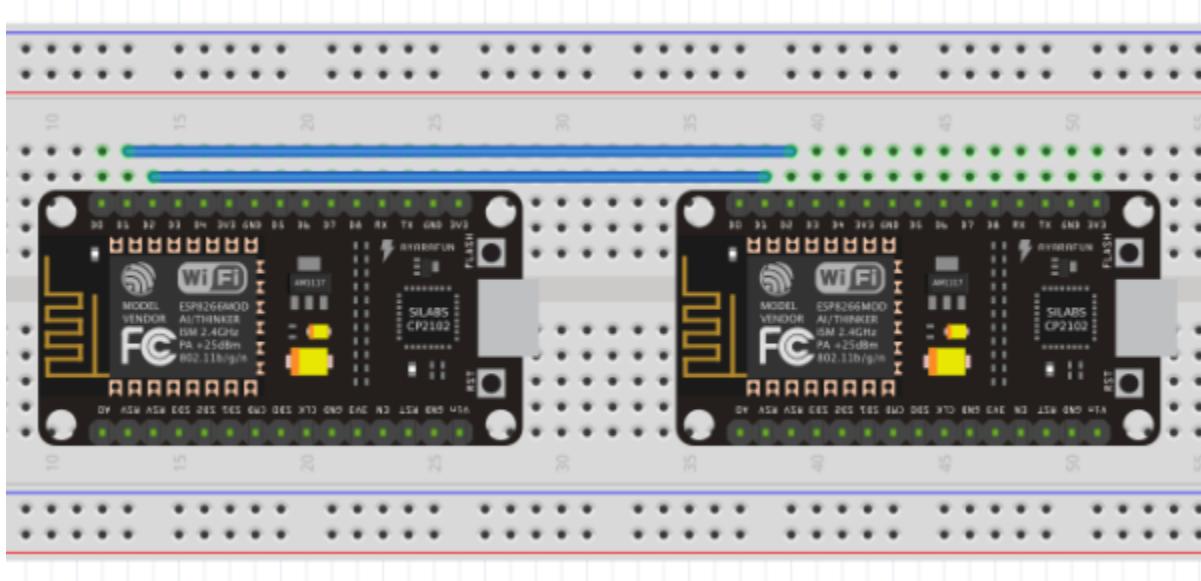
ผ่านมาสองตอนเรียบเรื่องกันเรื่องของ ESP8266/ESP32 ในการนำมาทำ Mesh Network ซึ่งข้ามาทำ ถ้าใครทำตามผ่านมาได้ ถึงตอนนี้ไม่มีอะไรยากเลยครับ ตอนนี้จะเป็นเรื่องของการเชื่อมต่อหรือ Bridge เข้า Local Mesh Network ของเราเพื่อออกไปสู่ Internet กันครับ ซึ่งในที่มีเราอาจจะเป็น Bridge Node ของเราระบีไปเพื่อต่อเข้ากับ Server Node แต่ก่อนอื่น เรามาดูเรื่องการส่งข้อมูลผ่าน Serial Port ของ Nodemcu ส่องตัวกันครับ เพราะสามารถประยุกต์ใช้งานอย่างอื่นได้อีก

ตัวอย่างเพื่อประยุกต์ใช้ในการส่งผ่านข้อมูลของ Nodemcu ส่องตัวผ่าน Serial Port

Redundant (Active-Standby): อย่างที่ทราบกันดีว่าพลังในการประมวลผลและหน่วยความจำของ ESP8266 นั้นมีจำกัด ซึ่งเราที่สามารถที่จะให้ Nodemcu ของเรางานสลับกันได้ กรณีตัวใดตัวหนึ่งตายไป อีกตัวก็เขียนมาทำงานแทนแล้วก็สั่งลัญญาณไป reset Nodemcu ตัวที่ตายไป พอดีตอนปลุกขึ้นมาใหม่ก็มีเวลาอยู่ใน Standby Mode กอย่างสั้นๆ ผ่านทาง Serial Port จะ ซึ่งเราสามารถทำ Heart Beat Check ผ่านทาง Serial Port ได้ โดยที่ Nodemcu สามารถทำงานแทนกันไปแทนกันมาได้ตลอด เพราะอย่างที่ทราบกันดีอุปกรณ์ดีขนาดไหนก็มีโอกาสที่จะ hang ได้ ถ้า reboot แล้วกับมาทำงานตามปกติได้ แต่ถ้าไม่ปกติอาจจะทำให้งานที่ทำอยู่เสียหายได้ ถ้าเป็นสถานที่ที่ต้องเดินทางก็คงลำบากอีก บางคราวอาจจะคิดว่าอ้าวว่า ถ้าเงินก็ให้มันทำงานสองตัวด้วยกันได้แล้ว ก็ทำงาพร้อมๆ กันเลย อันนี้อาจต้องมองแล้วแต่ค่าครับ เพราะบางที่เราอาจจะไม่ได้ใช้งานเป็น Sensor Node อย่างเดียว

Communication Channel: เคสีนี้จะเป็นเคสที่เราจะใช้กันในตอนที่ 3 นี้ครับ ซึ่งบางโปรเจกต์ที่ทำกับ Arduino หรือ MCU บางตัวก่อนหน้านี้ที่ไม่มี WiFi รองรับแล้ววันนึงของากจะส่งค่าอุณหภูมิเพื่อ monitor ผ่านทาง Internet ก็สามารถทำได้โดยที่ MCU เดิม ส่งค่าผ่านมาทาง Serial Port และใช้ ESP8266 รับค่า จากนั้นก็ส่งข้อมูลผ่าน WiFi

น่าจะพอเห็นภาพการใช้งานเพื่อสื่อสารระหว่าง Nodemcu 2 ตัวผ่าน Serial Port กันแล้ว เรา มาดูในส่วนของ Code กันบ้างซึ่ง เนื่องจากข้อจำกัดของ 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 เพื่อให้เที่ยวกันภาระน้ำหนักที่ตัวอย่างนี้กันครับ โดยใช้ mcu2 ตัว



เริ่มจาก Copy Code ด้านล่างนี้แล้วก็ Flash ลง Nodemcu ทั้งสองตัวเลย ซึ่งใน Code ผมกำหนดให้ D1 เป็น RX ส่วน D2 เป็น TX จะนั่นการต่อสายจะเป็นลักษณะไขว้กันระหว่าง 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);

}

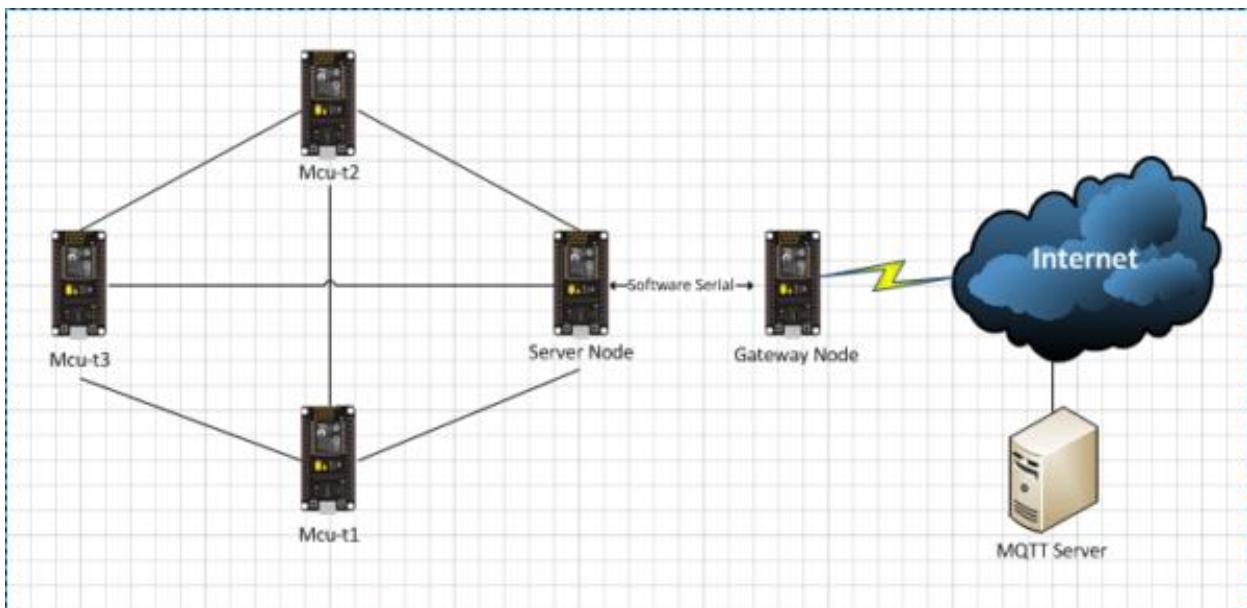
```

เมื่อเปิด Serial Monitor มาดูก็จะเห็นว่าค่าที่ print ออกมาต่อหน้า Serial Port(HW Serial) นั้นคือค่าที่เราอ่านได้จาก swSerial (Software Serial) ซึ่งวนลูปเพิ่มค่า i ไปเรื่อยๆ โดยเป็นค่าที่รับมาจาก Nodemcu อีกด้วยหนึ่ง ค่าที่ได้รับมานี้ก็ขึ้นอยู่กับเรา่ว่าจะเอาไปทำอะไรต่อ

```
>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 ของเรากันดีกว่า นั่น ก็คือการ Bridge ระหว่าง Local Mesh Network ของเรา เพื่อ ส่งข้อมูลออกไปยังโลกภายนอก กัน โดยที่ Scenario ที่เราจะ 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 แล้วส่งค่าออกผ่านทาง internet
- Gateway Node ที่รับข้อมูลจาก Server Node ผ่านทาง software serial จะส่งค่าไปที่ MQTT Server

*MQTT คืออะไร โดยละเอียด ไว้ติดตามกันต่อตอนที่ 5 แต่คร่าวๆ คือ messaging protocol ตัวหนึ่ง ซึ่งใช้รับส่งข้อความผ่าน IP Network อารมณ์คล้ายๆ Instant Message ซึ่งการรับส่งข้อความ จะใช้กิมมิลของการ Public(ส่ง)-Subscribe(รับ) กับ Topic ที่เราต้องการ ซึ่ง MQTT จะใช้มากในงานของ iot เพราะไม่ซับซ้อน กิน resource น้อยและค่อนข้าง realtime

ในส่วนของ **Code** กีประกอบด้วย 3 ส่วนด้วยกัน **Code** ชุดแรกสำหรับ Nodemcu ที่ต่อกับ 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 ซึ่งจุดที่จะเพิ่มก็คือการเรียกและกำหนดเวลาที่จะใช้งาน 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 ซึ่งในความเป็นจริงแล้วปลายทางของ Gateway ในการส่งผ่านข้อมูลไปจะเป็นอะไรก็ได้ อาจจะเป็น Webservice, Blynk, Firebase หรืออะไรก็ตาม ใช้หลักการเดียวกันเลย **Code** ที่ใช้สำหรับ Gateway Node เพื่อส่งข้อมูลไปยัง MQTT server ซึ่งถ้าไครอยากจะทดลองในส่วนนี้ติดตามจากตอนที่ 5 ในเรื่องของการตั้ง MQTT Server บน Google Cloud หรืออาจจะใช้บริการ [Cloud MQTT](#) ซึ่งมีส่วนให้บริการฟรีๆได้รับ

```

#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 และรอฟังว่าใน Mesh Network นี้มี Server Node นั้น ถ้ามีก็จะทำการบันทึก Server NodeID และสั่งจากนั้นก็ส่งเป็นแบบบัญญาช่างไปที่ Server Node ละ
- Step นี้ที่จะเพิ่เข้ามาก็คือ ที่ Server Node เมื่อได้รับค่าที่ส่งมาภายใน Mesh Network ก็จะทำการ Forward ต่อไปยัง Gateway Node ผ่านทาง software serial port ได้อะไรมีก็ส่งไปอ่านนั้น ให้ดู code ในส่วนของ receivedCallback ที่จะทำการ print ค่าออกรายทาง software serial


```
swSerial.print(msg.c_str());
```
- ที่ Gateway Node เมื่อได้รับค่าที่ส่งมาจาก software serial port ก็จะทำการส่งต่อค่านี้ไปยัง mqtt server

```

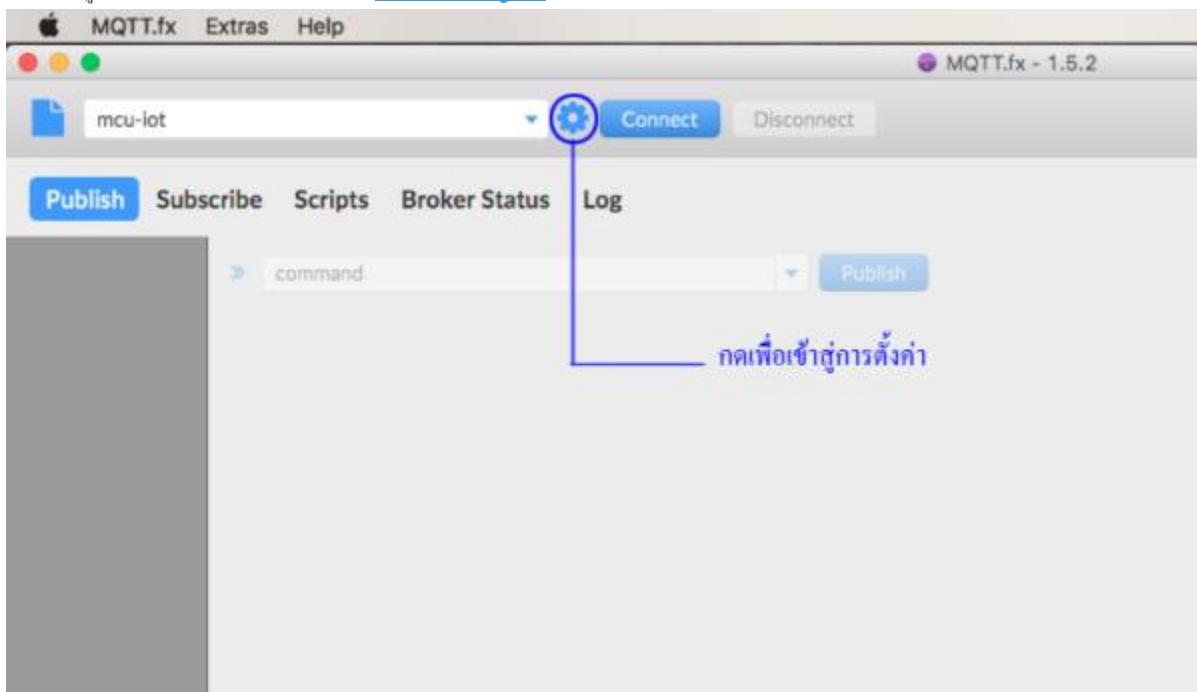
[{"nodename": "mcu-t1", "NodeID": 3896710322, "Temp": "25.90C", "Humidity": "43.80%"}, {"nodename": "mcu-t3", "NodeID": 3896054259, "Temp": "26.71C", "Humidity": "42.61%"}, {"nodename": "mcu-t2", "NodeID": 3896712201, "Temp": "25.70C", "Humidity": "42.50%"}, {"nodename": "mcu-t1", "NodeID": 3896710322, "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%"}]

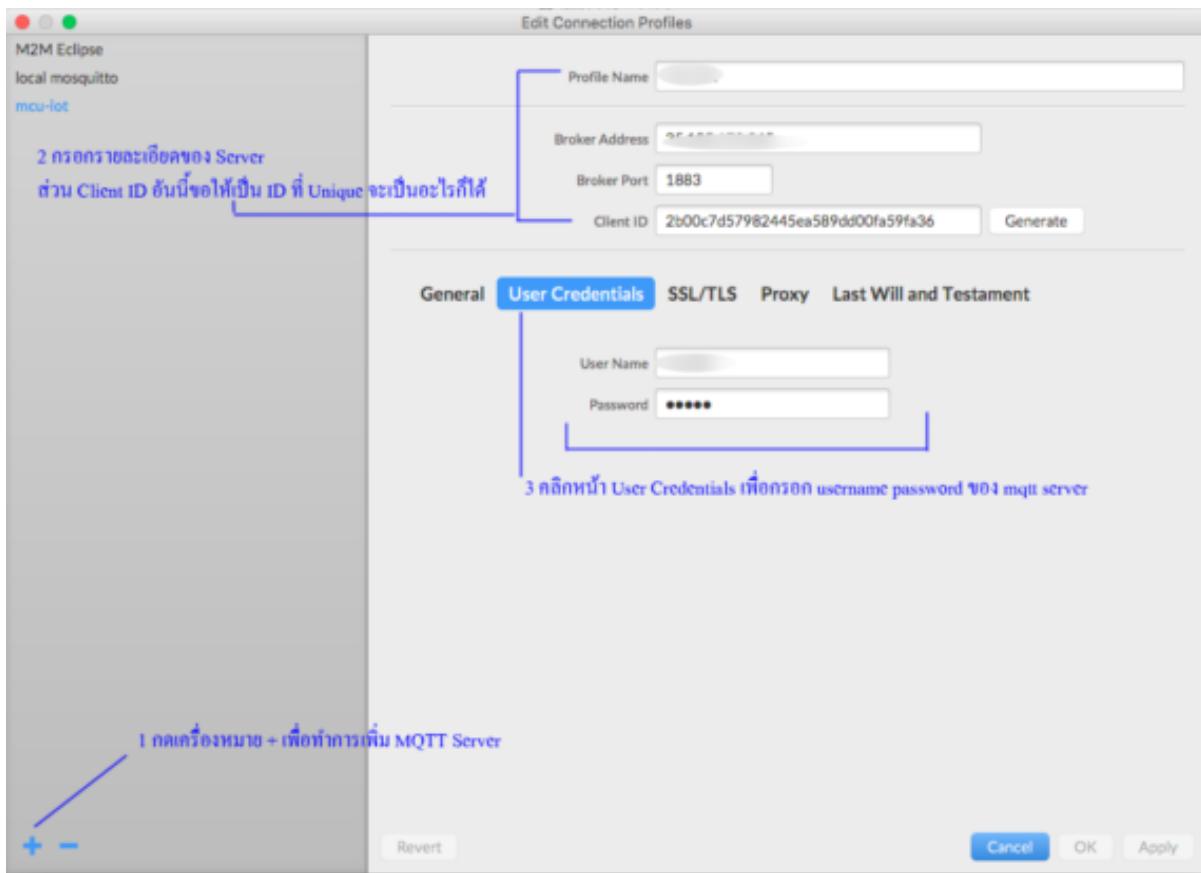
```

ข้อมูลที่รับเข้ามาจาก Software Serial Port จาก Gateway Node

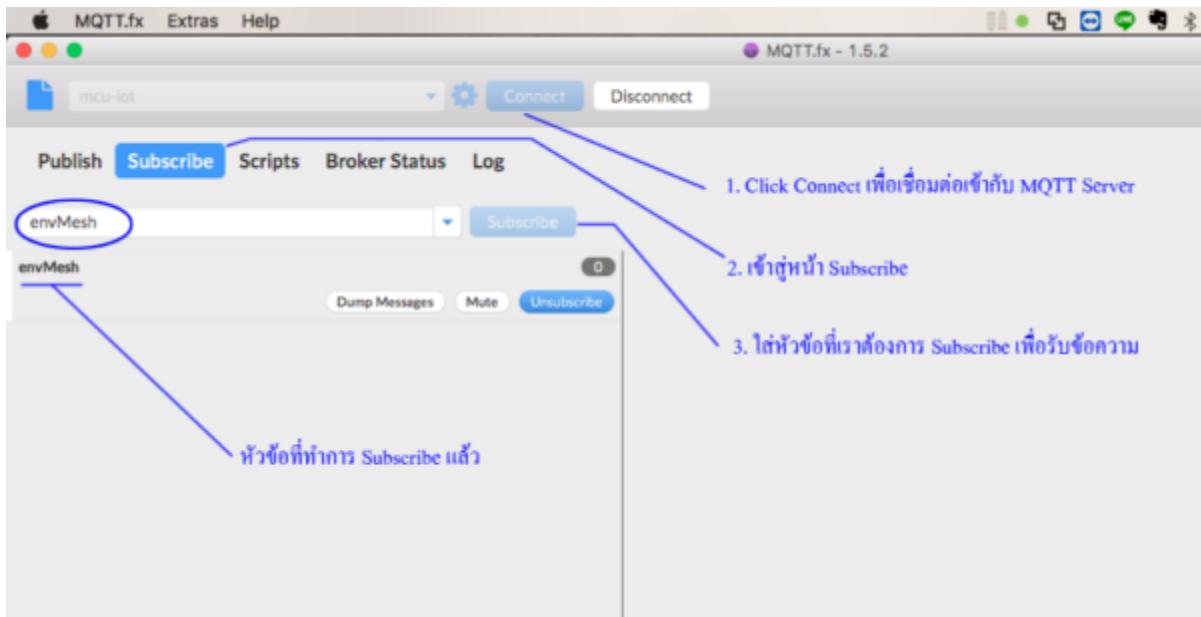
ส่วนผลลัพท์ที่อ่านค่าได้จาก MQTT Server นั้น мыใช้โปรแกรม [MQTT.fx](#) ซึ่งเป็น Client ที่มีให้ใช้งานได้หลากหลาย platform เลยเนื่องจากว่าเป็น Java based ก็สามารถโหลดได้จากเวปของ [MQTT.fx](#) ได้เลยครับ

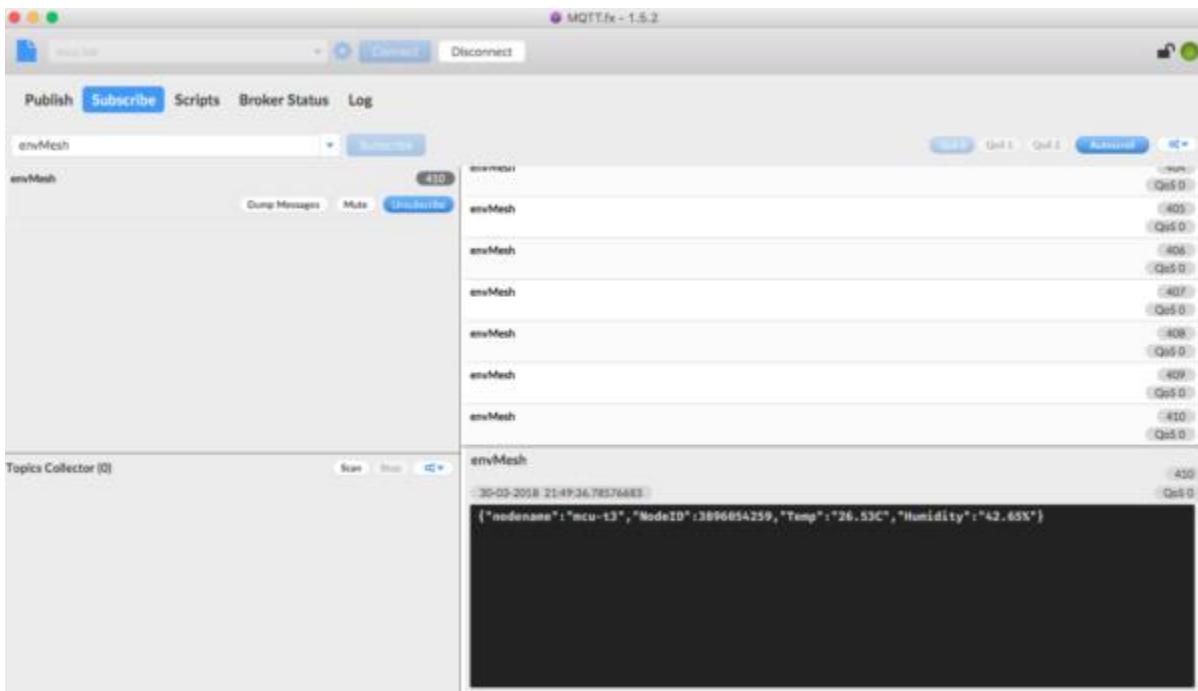
การใช้งานก็เริ่มจากการตั้งค่ากันก่อน ที่หน้าแรกให้คลิกที่รูปเปื้อง เพื่อเข้าสู่หน้าของการตั้งค่า แล้วก็ตั้งค่าตาม MQTT Server ของแต่ละท่าน ที่ใช้กันอยู่เลยครับ หรือจะใช้บริการฟรีของ [Cloud MQTT](#) ก็ได้



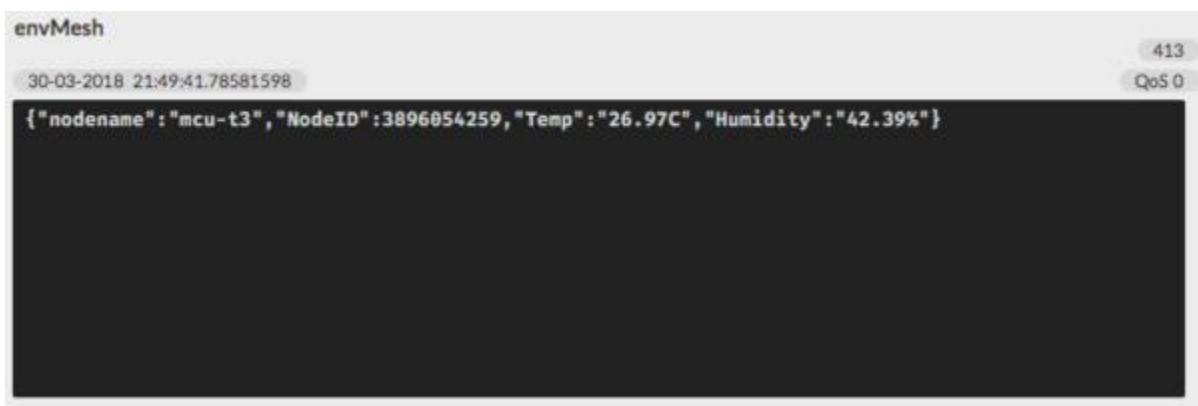


ทำการ Subscribe หัวข้อ envMesh ตาม Code ที่อยู่ใน Gateway Node





ข้อมูลที่ได้รับมาจะแสดงอยู่ที่ฟันที่สีดำ พร้อมกับ Timestamp



จากนั้นก็จะเห็นข้อความที่อ่านได้จาก MQTT Server ก็จะเป็นรูปแบบ JSON โดยส่วนมากจะมาจาก Local Mesh Network → Server Node ภายใน Mesh Network → Gateway Node ผ่านการ Bridge โดยใช้ Software Serial จากนั้นถึงต่อไปที่ MQTT Server ผ่าน WiFi/Internet → แล้วเราที่ใช้โปรแกรม MQTT.fx เข้าไป Subscribe เพื่ออ่านค่าที่ส่งมาให้หัวข้อ envMesh

จริงๆ สำหรับวันนี้ ใจความหลักเลยจะเป็นเรื่องของการใช้งาน Software Serial ที่ใช้ในการสื่อสารกันระหว่าง Nodemcu 2 ด้วย เรื่อง MQTT Server นั้นก็เป็นการทึ่งปมเพื่อให้ไปค้นกันต่อ หรือจะรอถึงตอนที่ 5 ก็ได้กับการ Setup MQTT Server ขึ้นมาใช้งานของบน Google Cloud และพบกันใหม่ตอนที่ 4 ซึ่งเราจะ Bridge ข้อมูลจาก Local Mesh Network ของเราผ่าน LORA Network กันครับ

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

บทความในตอนนี้ก็จะใช้เวลาเขียนนานหน่อย เนื่องด้วยงานที่ทำ แล้วก็อุปกรณ์เจ้ากรรม Heltec ESP32+OLED 915MHz ที่สั่งมา ต้นเดียวกับที่มี ที่เล่ายังนั่น ว่ารออุปกรณ์มาให้ครบเพื่อทดสอบ ซึ่งจากประสบการณ์ที่สั่งของจากนิ่มมาก่อน กรณีที่ไม่ได้สั่งแบบนำเข้ามาทดสอบ แบบเป็นทางการ ก็ควรสั่งเข้ามาอย่างน้อย 2-3 ตัวด้วยกัน เพื่อตัวหนึ่งใช้งานไม่ได้จะได้ทดสอบอีกด้วยหนึ่ง จะได้รู้ว่าอีมั้นเป็นที่ code ของเรา หรือว่าเป็นที่อุปกรณ์กันแน่ ซึ่งสุดท้ายด้านหลัง OLED เสีย อีกด้านนึง LoRa Module เสีย จนกันกันย่าน 915MHz



Heltec ESP32 SX1278 ความถี่ 433.175MHz ทดลองระยะประมาณ 80 เมตร ตัวส่งอยู่ในบ้าน ตัวรับอยู่ที่บ้านของเรา

เนื้อหาหลักๆ ในตอนนี้ก็จะเกี่ยวกับเรื่องของ LPWAN (Low Power Wide Area Network) อย่าง LoRa ทั้งการใช้งาน ข้อดีข้อเสีย กฎหมายที่เกี่ยวข้องในการนำมาใช้งาน เพราะถ้าติดตามอ่านมาตั้งแต่ตอนที่ 1 ([ESP8266/ESP32: Introduction & Painlessmesh](#)) ใจความหลักในการใช้งาน Mesh Network ด้วย Painlessmesh นั้นง่ายไปตั้งแต่ตอนที่ 3 แล้วรับที่ เหลือเป็นการประยุกต์มากกว่า ว่าเราจะใช้งาน Mesh Network ให้เหมาะสมยังไง หรือเพื่อจุดประสงค์ไหน เช่น

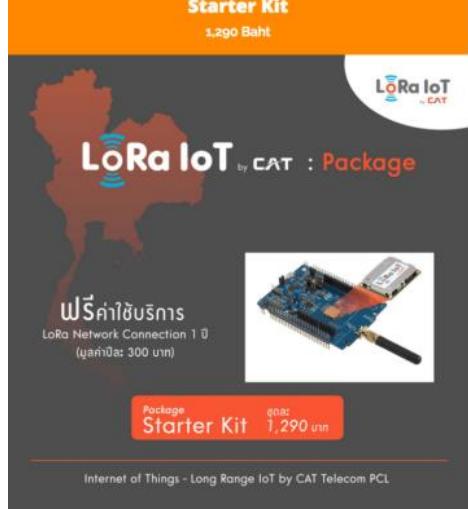
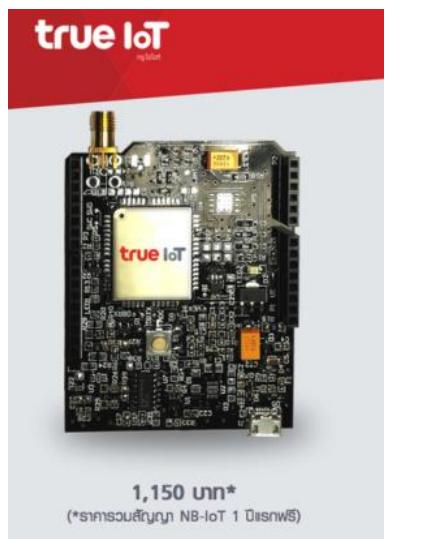
- เพื่อลด Single Point of Failure
- เพื่อขยายพื้นที่การครอบคลุมในการส่งข้อมูล
- ลดต้นทุนในการขยายอุปกรณ์กระจายสัญญาณ

จากนั้นการส่งข้อมูลจาก Mesh Network ซึ่งเปรียบเหมือน Local Network ของเรา จะต้อง Bridge ยังไงเพื่อให้ข้อมูลเหล่านั้น ไปยัง Server, Database, Cloud, MQTT Broker หรืออะไรก็แล้วแต่ที่อยู่ Network อีกหนึ่งหรือผ่าน Internet ซึ่งก็จะได้ทดสอบทำกันไปในตอนที่ 3 ([ESP8266/ESP32: Painlessmesh Bridge](#)) ซึ่งถ้าเราจะไม่ใช้ Software Serial ในการ Bridge ก็ยังสามารถทำได้อีกด้วยวิธี ซึ่งวันนี้เราจะมา Bridge ผ่าน LoRa กัน

LoRa vs LoRaWan

ผู้ให้บริการทางด้านโทรศัพท์เคลื่อนที่ AIS ([AIS NB-IoT](#)) และ True ([True IoT](#)) ก็เปิดตัวบริการ NB-IoT ออกมาน่า ส่วน CAT ก็มาทางสาย LoRa ([LoRa IoT by CAT](#)) ซึ่งทั้งคู่ก็ออกแบบเพื่อรองรับการสื่อสารสำหรับบริการประเภท IoT ใน

ส่วนของ AIS และ True ที่เป็นผู้ให้บริการมือถืออยู่แล้ว มีความต้องการใช้ในการจัดสรรงานบริการ LTE การมาให้บริการ NB-IoT ก็จะประยุกต์ด้านทุนไปได้ยอดเยี่ยมเป็นการ Utilize ความต้องการที่ตัวเองได้รับอนุญาตมาด้วย เท่าที่เห็นก็เป็น 3 ผู้ให้บริการหลักๆที่ปิดตัวออกมากันชั่งหัวข้อของเราในวันนี้ก็จะคุยกันในเรื่องของ LoRa และการใช้งานร่วมกัน Mesh Network จะนั่นหมายความว่าต้องต่อตัวกันระหว่าง LoRa และ LoRaWan กันก่อน

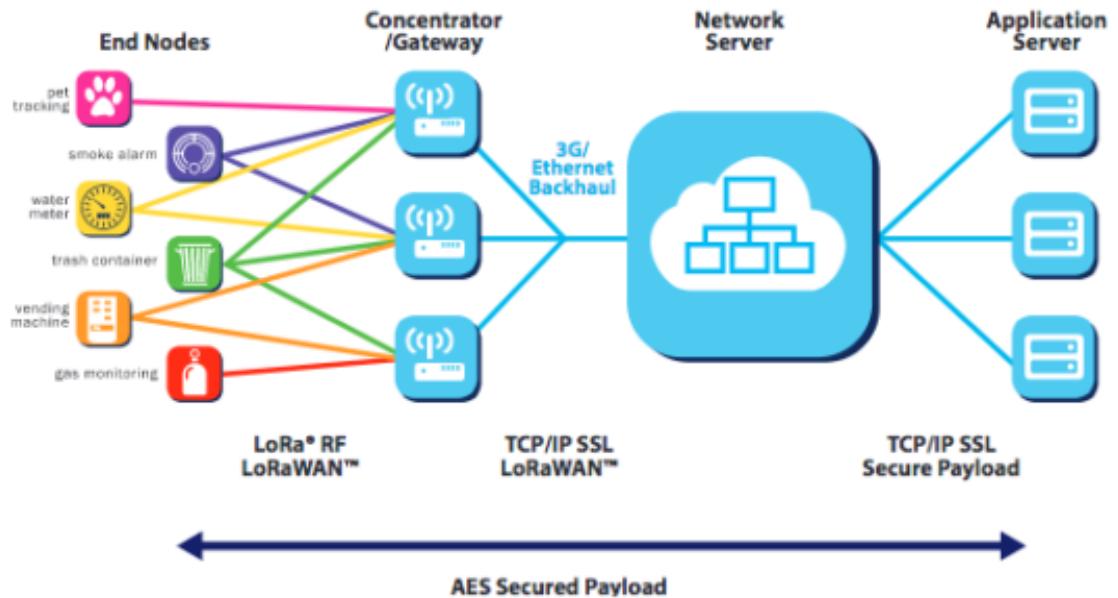


LoRa จะเป็นการสื่อสารในระดับ Physical Layer ซึ่งหมายความว่าจะใช้งานในลักษณะแบบ P2P ไม่มี Encryption ส่งข้อมูลกันแบบ Broadcast ออกไปโดย ฉะนั้นถ้ามีคนตั้ง LoRa Node ขึ้นมาแล้วใช้ค่า Setting เดียวกันเราจะสามารถที่จะเห็นข้อมูลได้ซึ่ง LoRa ถูกพัฒนาเป็น Wireless Data Communication โดยบริษัท Cycleo (Grenoble, France) จากนั้นผู้ผลิตชิป Semtech ที่เข้าไปร่วมบริษัทนี้มาในปี 2012 ถูกคุณสมบัติของชิปอย่าง SX1276 ที่รองรับย่านความถี่ 920-925MHz ตามประกาศของ กสทช. ที่จะเห็นว่าที่ LoRa นั้นส่งได้ไกลนั้นส่วนนึงก็ เพราะ High sensitivity ที่รองรับถึง -148dBm เลย ([ข้อมูล Semtech SX1276](#)) แต่ใช้งานจริงจะได้เท่าไหร่ต้องดูกัน



LoRa module รุ่นต่างๆ แฉะนนเป็น RA-02 จาก Ai-Thinker ที่ใช้ชิป Semtech SX1278/SX1276 สำหรับความถี่ 433MHz และส่องตัวล่างสำหรับย่าน 915MHz ส่วนส่องตัวล่างขวาเป็น RM95W จากค่าย Hope RF สำหรับความถี่ย่าน 915MHz

ส่วน LoRaWan นั้นจะอยู่ On top ของการสื่อสารแบบ LoRa Radio อีกทีซึ่งจะมีในส่วนของ Protocol การรับส่งข้อมูลเข้ามา รองรับ Encryption ซึ่งใช้ AES-128 ในการเข้ารหัส สามารถทำ QOS ได้ด้วย โดยรายละเอียดของ LoRaWan นั้นจะมีอยู่ 3 ค่อนข้างเย่อพอสมควร รวมถึงในส่วนของ Device End Node ก็ยังสามารถจำแนกได้ออกเป็น Class A B C อีก และการทำงานก็จะแบ่งออกเป็น Tier ตามรูปคือล่างนี้เลยครับ



จะนั้นการจะเลือกใช้ LoRa หรือ LoRaWan ก็ควรเลือกให้เหมาะสมกับ Application ที่จะใช้งานด้วย และที่สำคัญควรอยู่ตามข้อกำหนดของ กสทช ด้วยครับเนื่องจากคลื่นความถี่เป็นทรัพยากร่วมรวมที่มีการใช้งานร่วมกัน ถ้าใช้ LoRa ก็ต้องดูในเรื่องของ Duty Cycle เองที่ว่าปีไม่ใช่สั่งข้อมูลตลอดเวลา ของการใช้งานคลื่นความถี่ตลอด รวมถึงกำลังส่งที่ถ้ามากไป ก็จะไปรบกวนการใช้งานของอุปกรณ์ที่ใช้ Application ที่ใช้งานในความถี่เดียวกัน

ข้อมูลเพิ่มเติมจาก LoRa Alliance ที่น่าศึกษาซึ่งทาง LoRa Alliance ที่เป็น Non Profit Organization ที่ผลักดันในการเรื่องของการใช้ LoRa/LoRaWan เพื่อใช้งาน IoT

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

Limitations

เราสามารถใช้ชุดข้อมูลที่ได้จากการใช้ LoRaWan กันบ้าง ไม่ใช่ว่าเทคโนโลยีที่ดีอย่าง LoRa/LoRaWan จะเหมาะสมสำหรับทุก Application ฉันได้พัฒนา Mesh Network ที่จะเหมาะสมกับ Application ในบางประเภท หรือบางงาน เพราะถ้าเราต้องการเอา LoRa/LoRaWan ไปส่งข้อมูลแบบ Realtime ที่คงไม่เหมาะสมนั่นจะกลายเป็นการจองซองลัญญาตลดเวลา หรือให้ส่งข้อมูลขนาดใหญ่ๆ ที่คงไม่เหมาะสมกัน เพราะถูกจำกัดด้วยข้อมูลที่ส่งขนาดไม่เกิน 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

เนื่องจากคลื่นความถี่อีกเป็นทรัพยากรที่ต้องถูกจัดสรรเพื่ามีอยู่อย่างจำกัด และต้องมีกำกับดูแลในเรื่องของการใช้งาน ในประเทศไทยก็จะมีองค์กรที่ทันสมัยว่ากๆ องค์กรหนึ่งซึ่งคือ กสทช ([ดำเนินกิจกรรมการกระจายเสียง กิจการโทรทัศน์ และกิจการโทรคมนาคมแห่งชาติ](#)) ที่เดียวๆ ก็จะชอบมีคนออกมาให้ข่าวว่า เคยวึดในอนุญาตมั่ง ปรับน้ำหนึ่นนั่นนั่ง แล้วสุดท้ายเรื่องนี้เงียบไป มาครู่เรื่องการใช้งานความถี่ของ ragazzi ก็กว่ารับจากข้อมูลที่หมายจากเวปของหน่วยงาน กสทช เองบอกได้เลยว่าเล่นอาจงที่เดียว บางทีคืนหาด้วยเลขความถี่ต้องใช้เลขไทย ไม่เจ็บไม่เจอ ซึ่งผู้รวมรวมมาได้ดังนี้โดยยังคงความถี่ย่านที่รวมจะเห็นคุณเคยเห็นการใช้งานกันสำหรับ LoRa, RFID และ RC นั้นก็คือ EU433 ISM Band, AS923-925MHz

- ตารางกำหนดคลื่นความถี่แห่งชาติ 2560 ([Download Link](#))
- กฎด้าน 2560: เอกสารประกอบการรับฟังความคิดเห็นสาธารณะ ร่างประกาศเกี่ยวกับการใช้คลื่นความถี่ 920-925MHz ([Download Link](#))
- สิงหาคม 2560: ร่างประกาศเกี่ยวกับการใช้คลื่นความถี่ 920-925MHz และแนวทางการอนุญาตเพื่อประกอบกิจการ IoT ([Download Link](#))
- พฤศจิกายน 2560: ราชกิจจานุเบกษา – หลักเกณฑ์การอนุญาตให้ใช้คลื่นความถี่ย่าน 920-925MHz ([Download Link](#))
- พฤศจิกายน 2560: ราชกิจจานุเบกษา – มาตรฐานทางเทคโนโลยีเครื่องโทรศัพท์เคลื่อนที่และอุปกรณ์สำหรับเครื่องวิทยุคมนาคมที่ไม่ใช่ประเภท RFID ซึ่งใช้คลื่นความถี่ย่าน 920-925MHz ([Download Link](#))
- มกราคม 2561: ราชกิจจานุเบกษา – หลักเกณฑ์และเงื่อนไขการอนุญาตให้ใช้คลื่นความถี่สำหรับอากาศยานซึ่งไม่มีนักบินสำหรับใช้งานเป็นการทั่วไป ([Download Link](#))
- กุมภาพันธ์ 2561 ราชกิจจานุเบกษา – เครื่องวิทยุคมนาคมและสถานีวิทยุคมนาคมที่ได้รับการยกเว้นไม่ต้องได้รับใบอนุญาตตามพระราชบัญญัติวิทยุคมนาคม ([Download Link](#))
- ความถี่วิทยุและใบอนุญาตวิทยุโทรศัพท์เคลื่อนที่และอุปกรณ์สำหรับใช้งานเป็นการทั่วไป ([Download Link](#))

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

จะนี้สำหรับส่วนของคลื่นความถี่ **433MHz** ซึ่งที่เราจัดทดสอบกันสำหรับ **LoRa** นั้น ก็จะไม่ได้เกี่ยวกับการใช้งานทั้ง **RFID** หรือการใช้งานในด้าน **RC** กันอាកเสยาน ไร้คนขับ แต่จะทดสอบโดยอาศัยความตามเครื่องวิทยุคมนาคมและสถานีวิทยุคมนาคมที่ได้รับการยกเว้นไม่ต้องได้รับใบอนุญาต ซึ่งย่าน **433MHz** นั้นห้ามส่งเกิน 10 มิลลิวัตต์

สำหรับย่าน **920-925MHz** นั้นซึ่งมีข้อกำหนดในส่วนของ มาตรฐานทางเทคนิคของเครื่องโตรคณานคมและอุปกรณ์ กสทช.มท. 1033 – 2560 ว่าด้วยเรื่องเครื่องวิทยุคมนาคมที่ไม่ใช่ประเภท **Radio Frequency Identification: RFID** ซึ่งใช้คลื่นความถี่ย่าน **920-925MHz** เครื่องเดียวกันด้วยกัน ไม่เกิน 50 มิลลิวัตต์ได้รับยกเว้นไม่ต้องได้รับใบอนุญาตให้ทำมิใช่นำเข้าและนำออกซึ่งเครื่องวิทยุคมนาคมและใบอนุญาตให้ดึงสถานีวิทยุคมนาคมแต่ไม่ได้รับยกเว้นใบอนุญาตให้ค้ำชั่งเครื่องวิทยุคมนาคม

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)

ถ้าขึ้นกันตาม **Frequency Plan** และตามช่วงความถี่ที่ กสทช ปลดล็อกให้กับอุปกรณ์และบริการ IoT ก็สามารถใช้ได้ส่องช่วงเลขครับ ทั้ง AS920 กับ AS923 ([The Things Network Frequency Plan](#))

จากข้อมูลด้านบนนี้ ที่ต้องอ่านกันดีๆ หลักๆ ก็คือรับข้อมูลความถี่ความถี่ที่ใช้งานและจุดประสงค์ที่ใช้งาน จากนั้นก็คุ้มหลักเกณฑ์และเงื่อนไขที่ประกาศในราชกิจจานุเบกษา และก็ยังต้องดูด้วยว่าในประกาศนี้มีการยกเลิกประกาศตัวก่อไว้หรือไม่ หรือพากประโยคที่ว่าเว้นแต่กำหนดเป็นอย่างอื่น พากร่างกับการรับฟังความคิดเห็นสาธารณะก่อนใช้ประกอบได้ กรณีที่สุดหากมีข้อสงสัยในเรื่องของการใช้งานความถี่ การผลิตหรือ

นำเข้าเรื่องอุปกรณ์ความคิดเห็นที่ใช้ความถี่ กีวารของความต้องจาก กสทช เป็นหลักซึ่งกีวารทำเป็นหนังสือแจ้งไปเพื่อให้ กสทช ตอบกลับว่าสามารถทำได้ หรือไม่สามารถทำได้ หรือให้ข้อมูลตามข้อกำหนดใหม่

Scenario

กรณีในบ้านที่เราสร้าง Bridge ผ่าน LoRa ซึ่งข้อดีของ LoRa เลยที่เกี่ยวกับ Power Consumption ที่ค่า และระยะในการส่งที่สูง คะแนนด้วยคุณสมบัตินี้ แทนที่จะต้องการเก็บข้อมูล หรือความคุณภาพนี้ จะต้องวาง Mesh Network ให้ครอบคลุมมากกว่าที่จะส่งข้อมูลออก Internet อาจจะไม่จำเป็น แต่สำหรับการส่งระยะไกลผ่าน LoRa มาแล้วค่าอัมพาต Internet จากผู้ของ LoRa Node ที่ทำตัวเป็น Gateway ที่ได้สำหรับตอนที่ 4 นี้จะเป็นการใช้งานผ่าน LoRa นะครับ แต่ถ้าใครอยาก Advance Setup LoRaWan Gateway และใช้งานผ่าน LoRaWan ที่ได้อ่านที่บอกไว้กับหลักการเดียวกันกับตอนที่ 3 ลองมาดูกัน

Scenario 1: Smart Farm

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

เทคโนโลยีใช้ Mesh Network หมายความว่า ได้ประযุชน์ในเรื่องของการครอบคลุมพื้นที่กว้าง จากการกระจายตัวของ Node และสร้างเป็น Mesh Network โดยที่จุดเชื่อมต่อสุดท้ายอาจอยู่ที่ขอบของ Mesh Network ก็จะเหมาะสมมาก สามารถลดต้นทุนในเรื่องของการสร้าง Network สำหรับการสื่อสารของแต่ละ Node ได้

Scenario 2: Falling Rock Alarm

สำหรับคนที่เป็นเกษตริกรที่ได้หว่าน ที่ใช้ในการตรวจสอบพื้นที่กลมจากภูเขา เนื่องจากได้หว่านนั้นมีสภาพภูมิประเทศเป็นภูเขา และอยู่ในเขตแผ่นดินไหว ซึ่งถนนบางเส้นนั้นก็ตัดผ่านภูเขาหนึ่งบนถนนทางภาคเหนือของประเทศไทย ซึ่งบางครั้งก็จะมีเหตุการณ์หล่นลงมา กว่าจะทราบเหตุก็ต้องมีคนโทรแจ้งจากที่ไปเจอ



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

ซึ่งที่ได้หัววินได้มีการติดตั้งเซนเซอร์ตรวจจับการสั่นสะเทือนของแนวคลื่นข่ายกันทินตกตามถนนไว้เพื่อตรวจจับว่ามีภัยรุ่งหล่นลงมา หรือมีคิมส์ไลล์ดองมาปิดถนนมั้ย โดยการใช้-camera สำหรับตรวจจับไม่ได้ตรวจดูตั้งแต่แนว เตาไฟบางส่วนที่ตัดผ่านภูเขา และต้องการส่งข้อมูลระยะไกล LoRa จึงตอบโจทย์ดี ขณะนี้เกิดลักษณะแบบนี้ เราสามารถเอา Mesh Network มาทำงานร่วมกับ LoRa ให้โดยที่ Sensor Node สามารถส่งเป็นบิรุณที่ก่อว่างเข็นได้โดยมีต้นทุนที่ต่ำ และส่งข้อมูล แจ้งต่อในไปที่ศูนย์เชิงเดือนด้านการ Bridge ผ่าน LoRa ที่อาจอยู่ไกลออกไปทางภาคใต้ ถ้าเมืองไทยไม่สามารถที่ก่อตัวมาเก็บรับ ปัญหาแล้วขับรถไปคดดูที่ซึ่งราษฎรช่วงหน้าฝน ก็จะถอนน้ำปืนไปรุ่งเสน่บ้างช่วงอุทก์เมืองกัน ถ้าเราสรุปว่างหน้าก็จะสามารถวางแผนการเดินทางหรือหลีกเลี่ยงเส้นทางได้ถูก

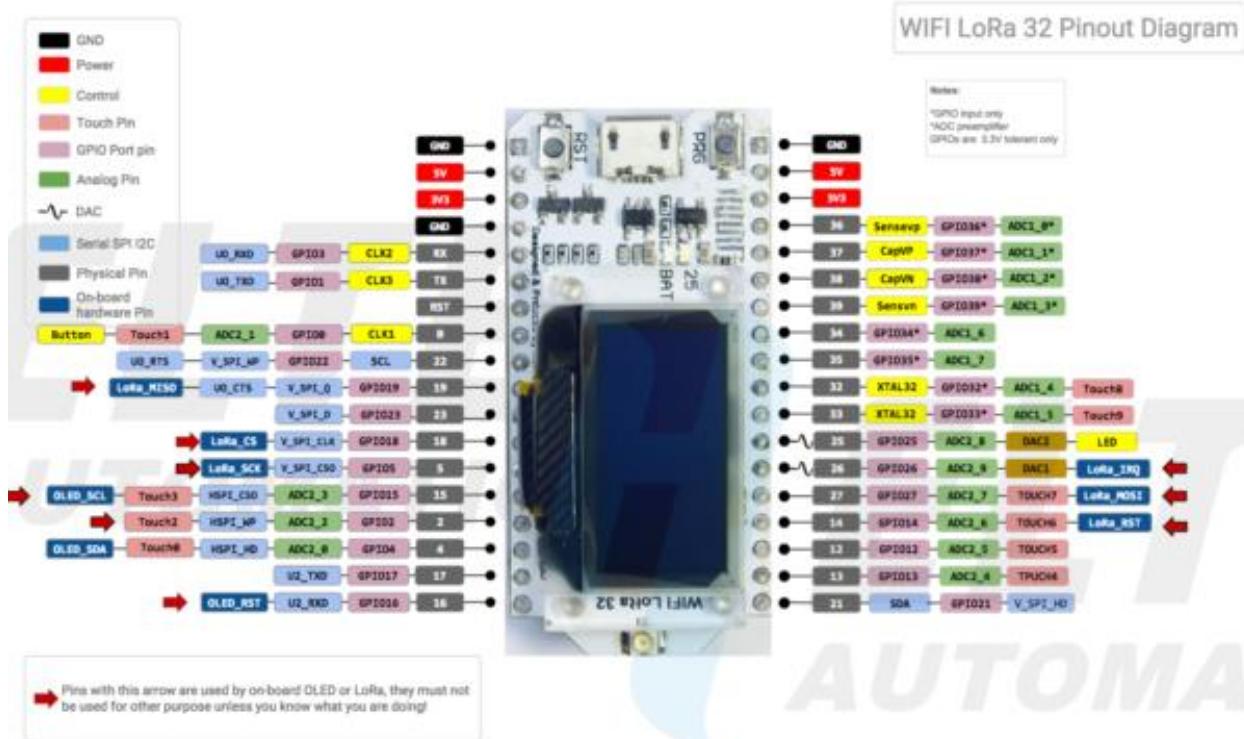
Scenario 3: Smart Bin

เกสนีก็เป็นแคสที่เกิดขึ้นจริงอีกเช่นกัน ลองคิดคุยกันว่าจะใช้จุดไหน ชุมชน ที่ต้องมีรถเก็บขยะเข้าไปเก็บทุกวันฯ หรือบางที่วันนึงอาจจะหลายรอบก็ได้ หรือบางที่เข้าไปเก็บ ขยะในถังอาจจะไม่มีเลขที่ได้

ซึ่งเกสนีก็มีคนทำเป็น Product Smart Bin ที่มีพื้นที่ห้าดองด้วยเพื่อเพิ่มพื้นที่การเก็บขยะได้มากขึ้น มีตัว Monitor ระดับของขยะเพื่อส่งคำสั่งไปแจ้งเดือนที่ศูนย์ว่า ให้มาเก็บขยะได้แล้ว รวมถึงจัดเส้นทางให้รถเก็บขยะได้ด้วย ซึ่งเทคโนโลยีการสื่อสารที่ใช้ได้มากก็จะเป็น LoRa หรือไม่ก็ Cellular (2G, 3G, NB-IoT) เพราะด้วยลักษณะการถังขยะของหมู่บ้าน หรือชุมชนในตปท อาจไม่เหมาะสมกับการนำ mesh มาใช้

Wiring & Diagram

ความจริงการใช้งานจะใช้งานผ่าน Module SX1278/1276 ของ Ai-Thinker ที่ได้แต่ต้องต่อสายเข้ากับ Breadboard หรือในส่วนของ RFM95W ก็ต้องมานักกรีกันอีก ขณะนั้นตอนนี้อาจยังติดสำหรับผม ก็ใช้อาร์ดูโอร์ด้าเร็วๆ ที่มาพร้อม ESP32+OLED ของ Heltec/TTGO เลยดีกว่า เพราะว่าทำการ wiring ไวยบน pcb ต้องขอyle แล้ว ซึ่ง pinout ก็จะเป็นดังรูปด้านล่างนี้



จะนั่งขาที่ใช้ในการกำหนดการใช้งานจะเป็นดังนี้ ซึ่งขาที่ถูกใช้ในการต่อ กับ module sx127x บนบอร์ดนี้ก็จะไม่สามารถนำไปใช้งานอีกอีกด้วย

```
#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 ที่ใช้ในการติดตั้งมีอยู่ 2 แหล่ง ด้วยกันคือ

จาก Espressif ผู้ผลิตชิป ESP32

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

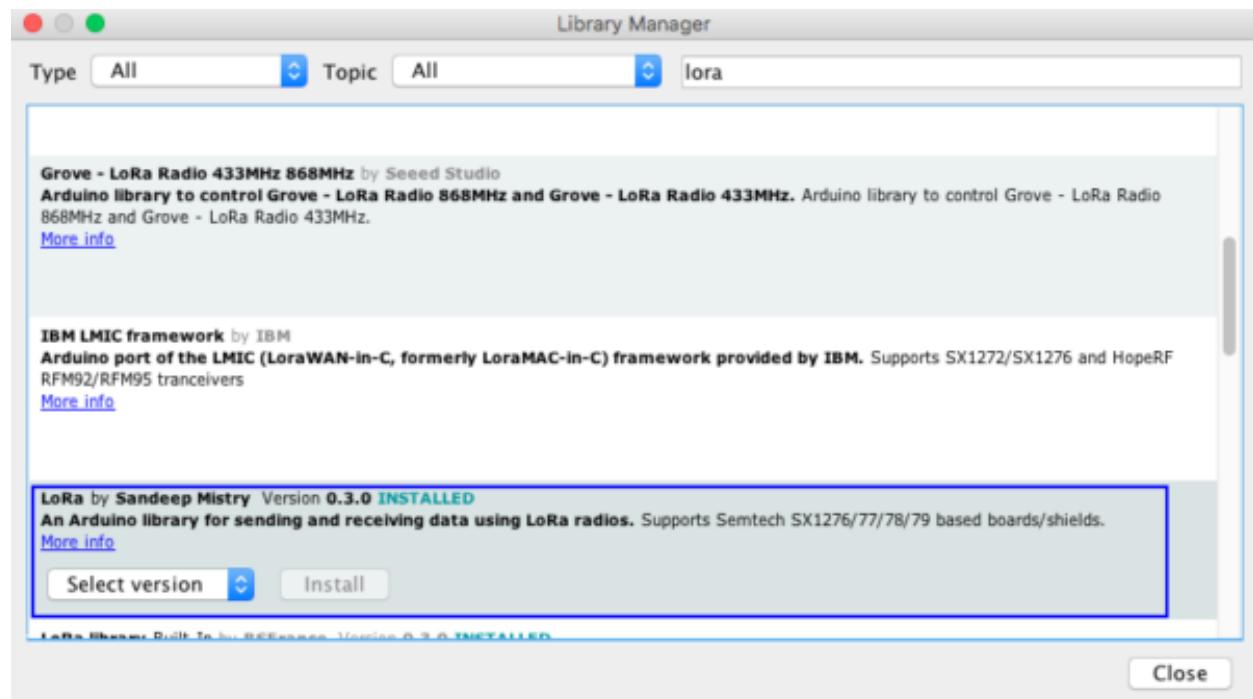
จาก Heltec ผู้ผลิตบอร์ด Heltec ESP32 LoRa

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

ข้อเดียวกันต่างระหว่างสองบอร์ดนี้คือ ถ้า Clone มาจาก Heltec จะมี Library LoRa มาด้วยซึ่งก็เป็นตัวที่ไม่มากจากของคุณ Sandeep Mistry แต่ถ้าการอัปเดตมีการ Update ก็จะไม่ถูก Update ผ่านทาง Library manager ต้องขอตามจากทาง Heltec เอง polymod เลือก Clone มาจากทาง Espressif แล้วทำการติดตั้ง LoRa Library เอง ส่วนวิธีการติดตั้งก็ขึ้นกับ OS ของแต่ละท่านที่ใช้อยู่ หลังจากติดตั้งไปเลือกทางไฟแล้วก็อ่าศัลจิกซึ่งขั้นบนนี้จะเหลือครับทำตาม Instruction ในแต่ละ OS ได้เลย

LoRa Library by Sandeep Mistry

สำหรับใครที่เลือกติดตั้ง Board โดยเลือก Source จาก Heltec ก็ข้ามขั้นตอนติดตั้ง Library นี้ไปได้เลยครับ แต่ถ้าใครที่ติดตั้งผ่าน Source ของ Espressif ก็ให้ติดตั้ง LoRa Library ของคุณ Sandeep Mistry โดยเข้าไปที่ Sketch → Include Library → Library Manager แล้วกันหากำกว่า LoRa ครับ จะเจอกายาด้วยกันให้เลือกตามรูปด้านล่างแล้วกด Install ได้เลยครับ



มาตรฐานในส่วนของ Code ที่จะใช้ทดสอบระยะกันบ้าง ตัวโปรแกรมก็จะแบ่งเป็นสองชุดคือฝั่งส่งและฝั่งรับ ฝั่งส่งก็ Count Number แล้วส่งออกไปเรื่อยๆ ส่วนฝั่งรับนั้นเนื่องด้วยไม่มี GPS Module ก็เลือกอ่าศัย GPS Stream จาก Blynk มาใช้ในการเก็บค่าพิกัดแทน เพื่อที่จะได้มาระยะเป็นอย่างไรบ้าง ซึ่งถ้าใช้บอร์ดที่ใช้ชิป SX1278 ในส่วนของ Band จะกำหนดเป็น 433.175MHz และถ้าเป็นบอร์ดที่ใช้ชิป SX1276 ในส่วนของ Band จะกำหนดเป็น 923.2MHz นอกจากนั้นไม่มีอะไรแตกต่าง ส่วนใหญ่ที่จะทดสอบโดยที่ไม่ได้สนใจเรื่องของ 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



กันที่จุด Start วางแผน โดยใช้ความถี่ในการรับ-ส่ง ที่ 433.175MHz ได้พิกัดมาเรียบร้อย

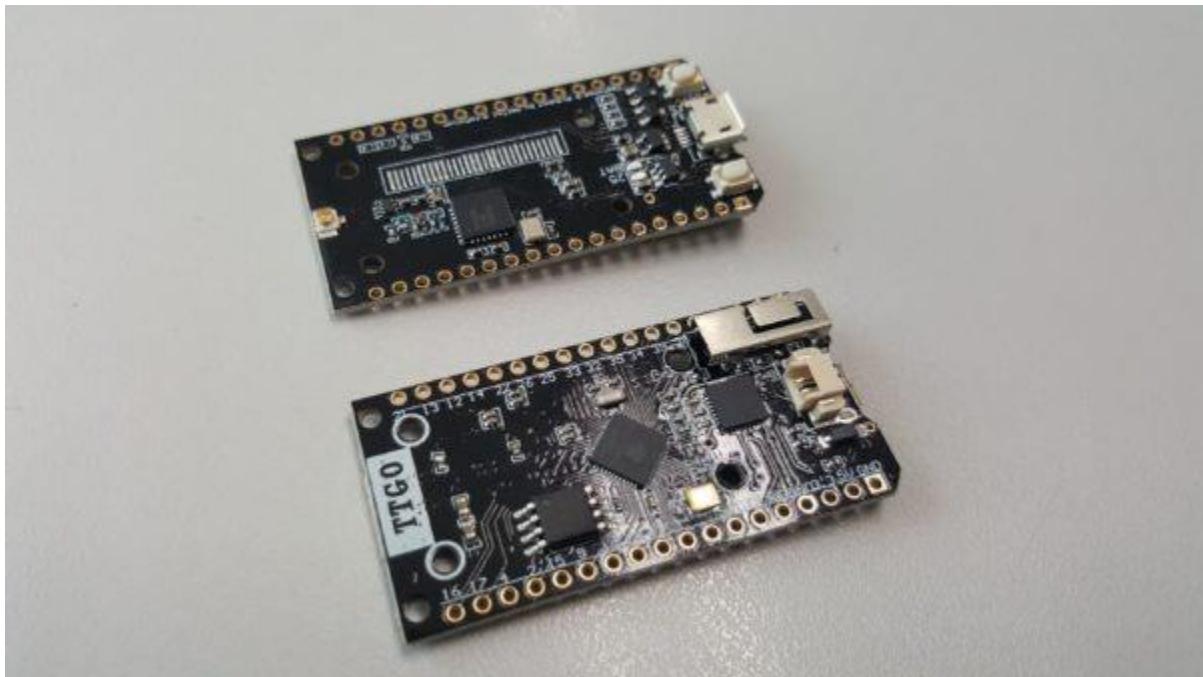


หลังจากนั้นกีเดินเล่นเลขครับ เล่นเอาหน่อยตกล่มีอกเหมือนกัน RSSI: -107dBm

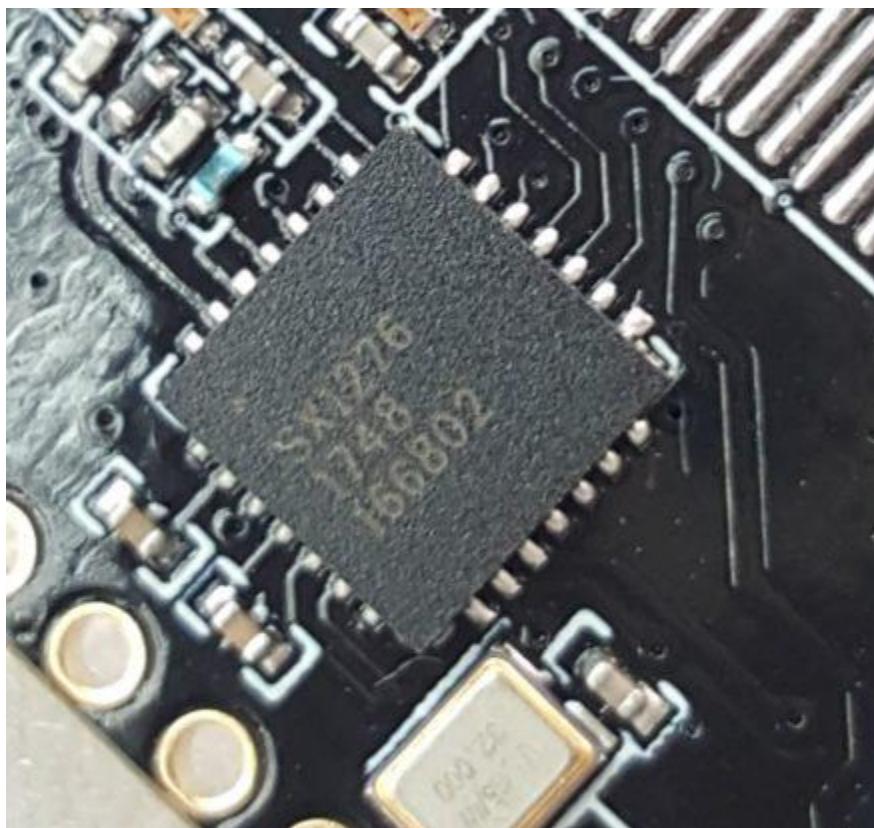


ต่ำสุดที่ Heltec ESP32 LoRa Sx1278 รับได้จะอยู่ที่ -108 dBm

ในความโฉคร้ายยังพอมีโชคดีอยู่บ้าง ถึงแม่ตัว Heltec ESP32 LoRa SX1276 ที่ใช้งานย่าน 915MHz จะเสียแต่ก็ได้บอร์ดใหม่มาทันพอให้ทดสอบ ซึ่งเป็นของ TTGO เป็น ESP32 LoRa SX1276 เหมือนกันเพียงแต่เปลี่ยนไม่มี OLED มาด้วยเท่านั้น ขณะนี้การวัดระยะ และค่า RSSI ก็เลือกษาอ่านค่า GPS จาก App Blynk บนโทรศัพท์มือถือแทน

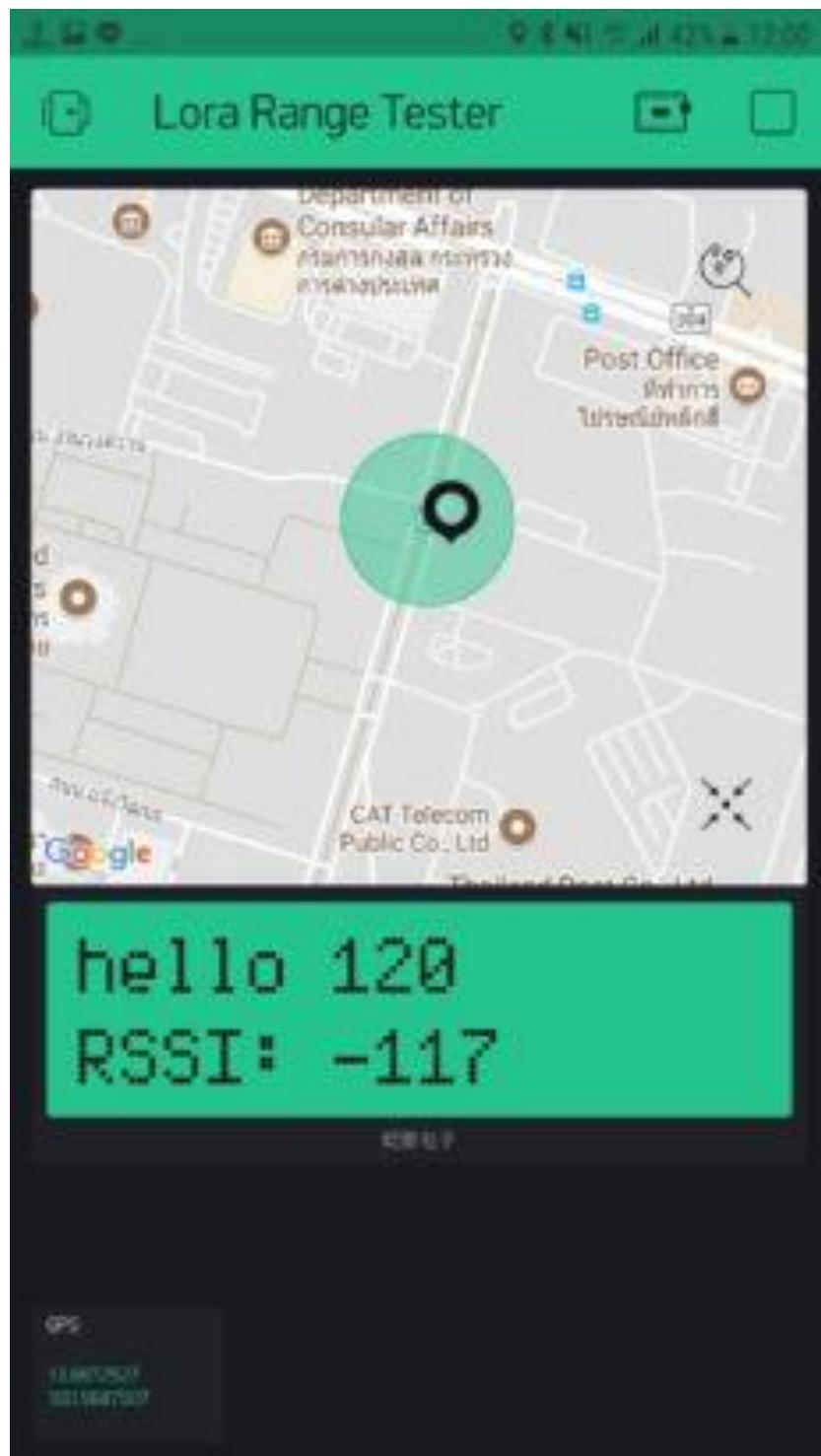


TTGO ESP32 LoRa SX1276

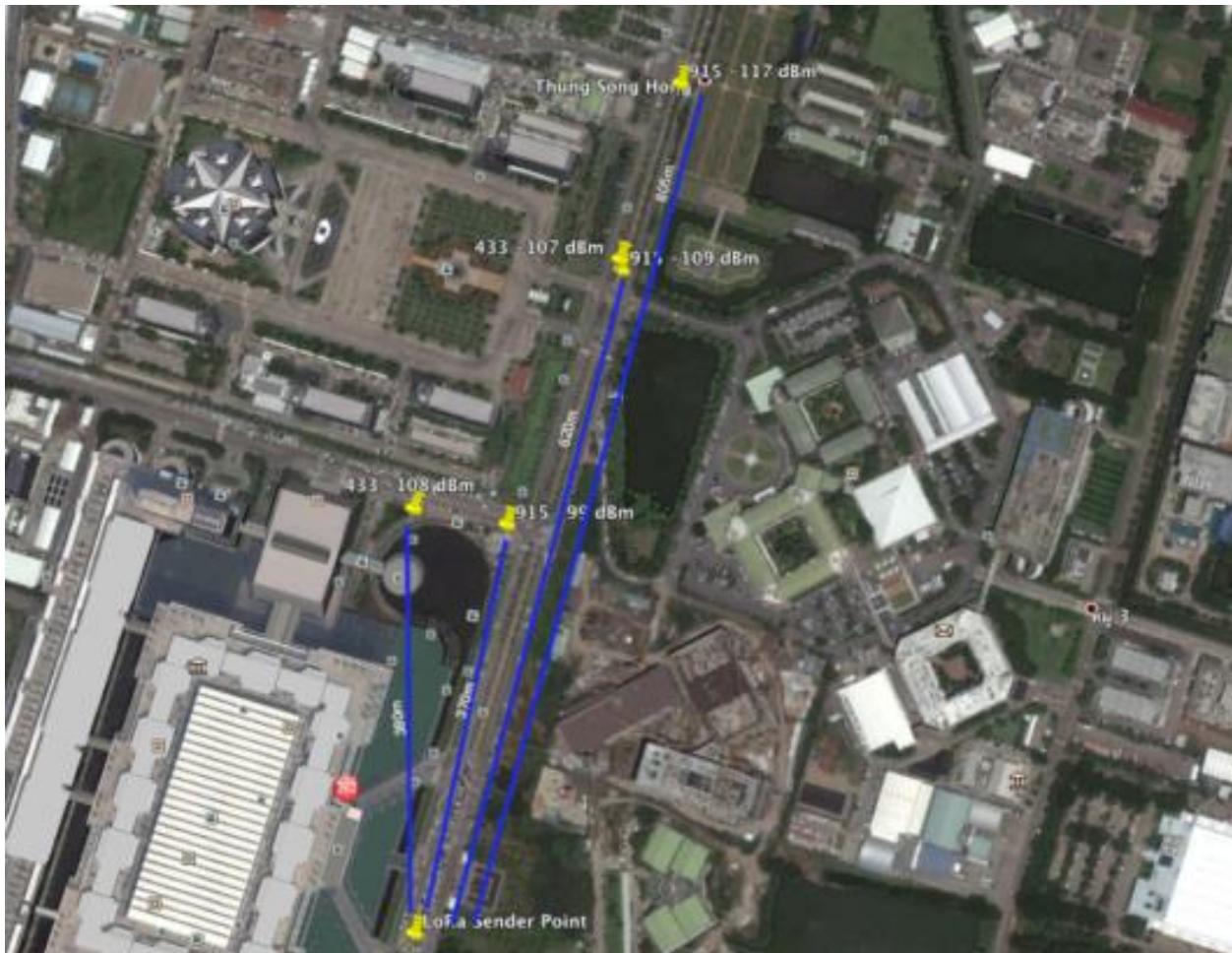


Semtech SX1276: LoRa ชิป ย่าน

915MHz



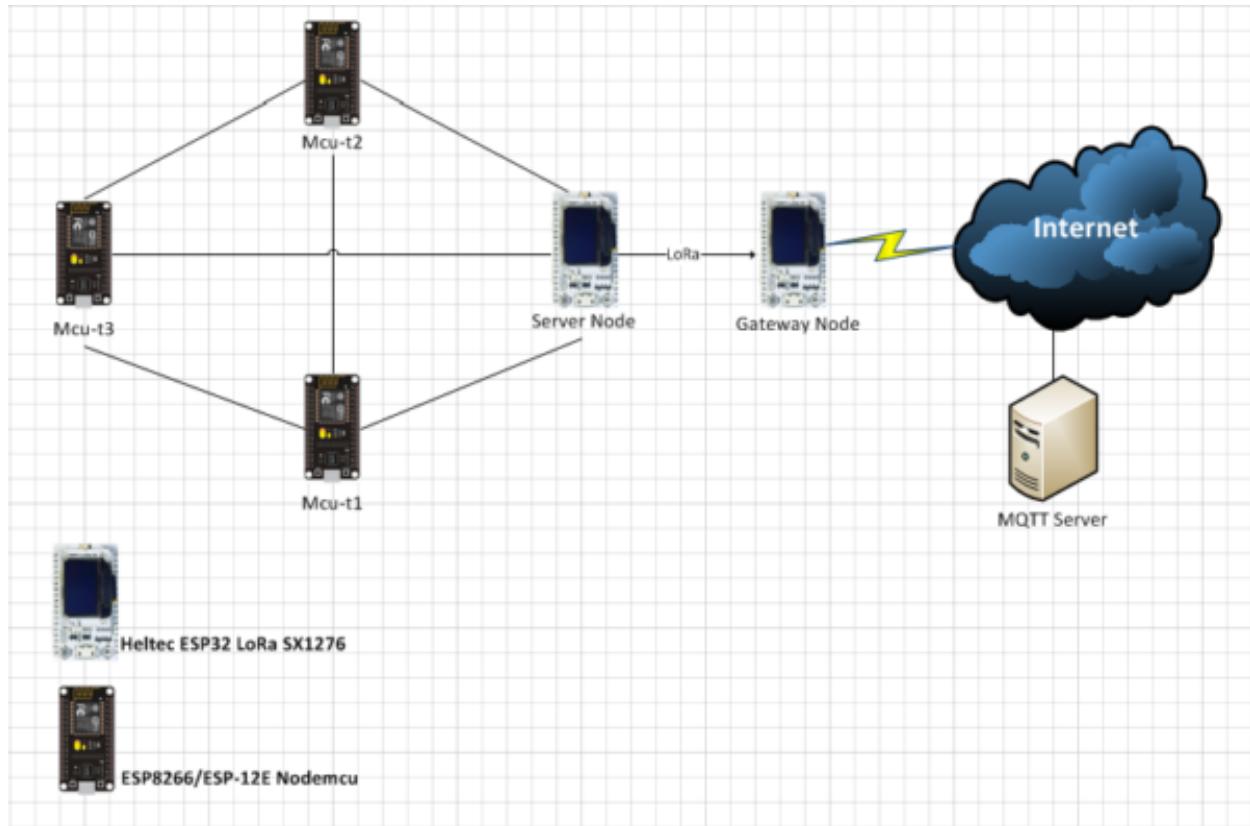
ชี้งการทดสอบสำหรับการรับ-ส่งที่ความถี่
923.2MHz ค่า RSSI ที่สามารถรับได้ต่ำสุดและรับได้ต่อเนื่องอยู่ที่ระหว่าง -110 ถึง -117 dBm



ผลการทดสอบระยะของแต่ละความถี่ที่ใช้ตามรูปด้านบนแลยกับ ชั้นก้าอาระยะหัวงผลไกลสุดซึ่งอยู่ใน Line of Sight ก็จะอยู่ที่ประมาณ 700-750 เมตรสำหรับการรับ-ส่งที่ความถี่ 923.2MHz ส่วนคลื่น 433.175MHz ระยะหัวงผลก็จะอยู่ที่ประมาณ 600 เมตร ทำให้การใช้งาน LoRa นั้นค่อนข้างสมชื่อ Long Range เลย เพราะนี่คือใช้สายอากาศที่แคมมา ถ้าใช้สายอากาศเดียว และตั้งเสาสูงน่าจะไปได้ไกกว่านี้ แต่ทั้งนี้ทั้งนั้นกำลังส่งก็ไม่ควรเกิน กสพช กำหนดนะครับ ชั้นเคลที่เขียนนี้ “สำหรับการทดลองเท่านั้น” เพราะจาก Code ได้มีการกำหนด TX Power ไว้ที่ 17dBm บวกกับเสาที่แคนมานำจะไม่เกิน +3dBm ติกลมๆกี 20dBm หรือ 100 milliwatts แล้ว (แต่ส่วนตัวคิดว่าไม่น่าจะถึง น่าจะมี loss เยอะ ถ้าอย่างไรก็จะต้องเอาอุปกรณ์มาวัด)

Painlessmesh Bridge with Lora

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



เพิ่งเห็นว่ามี Dependency Library เพิ่มเข้ามาสำหรับ Painlessmesh อีกด้วยคือ 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 ก็ส่งต่อค่ามาไปยัง 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 ที่ Domain ของ 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 = "";

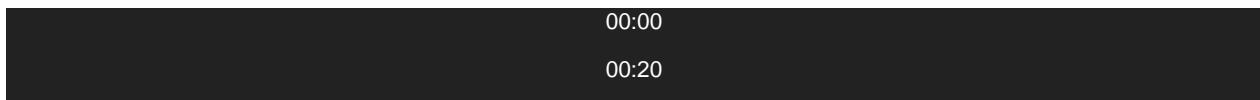
}

```

ผลลัพธ์ที่ได้
 ถ้าทำตามขั้นตอนที่่าว่ามาทั้งหมดได้ 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 เมื่อได้รับข้อมูลจาก LoRa ก็จะเอาข้อมูลนั้น Publish ต่อไปยัง MQTT Server ที่เราได้สร้างกันไว้ตั้งแต่ตอนที่ 3.5 ถ้าคุณเคยใช้ MQTT ของเห็นว่าระยะเวลาที่ใช้นั้นค่อนข้างเร็วมาก ซึ่งหลังจากนี้การจะนำข้อมูลจากภายใน Mesh Network ไปใช้งานนั้ง่ายแล้ว จะเพียงลง Time Series Database เพื่อเอาไปแสดงผล หรือวิเคราะห์ข้อมูลต่อ ก็ไม่ยากแล้ว

สรุป

เนื้อหาในตอนนี้จะเน้นในเรื่องของ Data Communication อิฐรูปแบบหนึ่งนั่นก็คือ LoRa ในการนำมายังงานร่วมกับ Mesh Network ซึ่งอย่างที่เกริ่นไปตอนแรกครับ ถ้าผ่านตอนที่ 3 มาแล้วก็ขึ้นอยู่กับเราว่าจะไป bridge กับอะไรเพื่อส่งต่อข้อมูลงานใน Mesh Network ไปยังอีก Network หนึ่ง ถ้าตัว Server Node ของเราต่อเข้ากับ Ethernet Module ENC28J60 ก็สามารถเป็น Bridge ในตัวมันเองได้โดยส่งข้อมูลผ่านสาย Lan ก็ได้ ดังนั้นที่เขียนมาหากายหลายตอนมากก็เพื่อ喻าให้เข้าใจหลักการนั้น ก่อน ที่เหลือก็จะขึ้นอยู่กับการประยุกต์ใช้ให้เหมาะสมนั่นเองครับ ใช่ว่าจำเป็นจะต้องใช้งาน Mesh Network สำหรับงาน IoT ทุกเคส หรือ ว่าจำเป็นต้องใช้งาน LoRa เพื่อให้ได้ระยะไกลๆ บางครั้งต้องพ่วงอุปกรณ์ของเรามาเข้ากับ GPRS/LTE Module ก็อาจจะทำให้ไปได้ไกล กว่าก็คงช่วยบนของญี่ปุ่นให้บริการมือถือได้เลยด้วยซ้ำ ใกล้กว่า NB-IoT ตอนนี้ด้วย

ส่วนในตอนหน้าที่น่าจะเป็นตอนสุดท้ายของ Series เรื่อง ESP8266/ESP32 กับ Painlessmesh ก็จะเป็นตอนปลิกย่อyle ระยะในตอนที่ 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

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

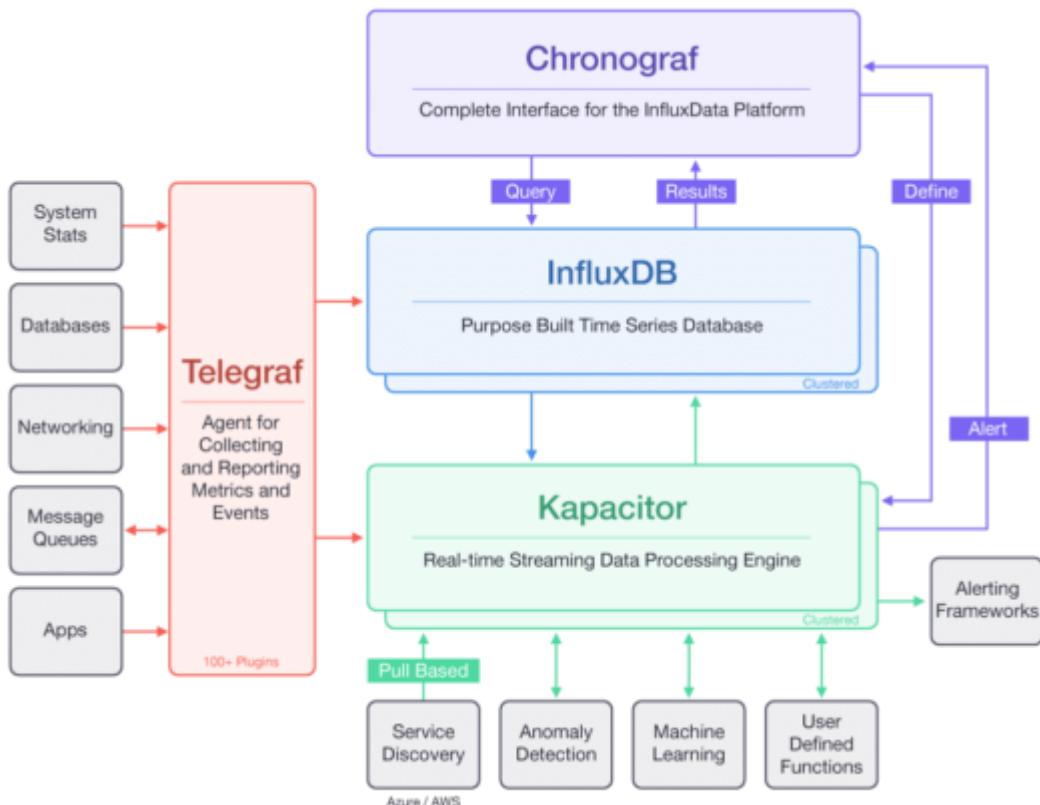
จริงๆเนื้อหาหลักของ Painlessmesh นั้นจะไปตั้งแต่ตอนที่ 3 แล้ว ที่เหลือจะเป็นลักษณะของการประยุกต์ใช้งานมากกว่า ไม่ว่าจะเป็นการ Bridge ด้วย LoRa หรือการติดตั้ง MQTT Server บน Google Cloud ซึ่งถ้าหากตอนล่าสุดตอนที่ 4 ถึงตอนนี้ก็ผ่านไป 4 เดือนละ ก็เลยกะมาจับ Series ของ Painlessmesh นั้นตามที่เคยเกริ่นไว้ว่าจะมีทั้งหมด 5 ตอนด้วยกัน โดยตอนนี้ก็จะเป็นเหมือนตอนแคมป์ลากาทางละ เพราะจะเป็นการนำข้อมูลต่างๆนั้นไปเก็บอยู่บน Time Series Database ที่ชื่อว่า InfluxDB และนำไปแสดงผลบน Dashboard ของ Grafana

ก่อนเข้าไปสู่การติดตั้ง InfluxDB ซึ่งเป็น Time Series Database (TSDB) โดยที่เป็นการเก็บข้อมูล เมมโมรี่เก็บ log ที่เก็บไปต่อๆเรียงตามวันที่และเวลาต่อๆกันไป เช่นข้อมูลราคาหุ้น ข้อมูลอุณหภูมิ ความชื้นในแต่ละช่วงเวลา ซึ่ง InfluxDB ก็เป็นหนึ่งใน TSDB ที่ถูก Optimize มาให้ทำงานในลักษณะนี้ โดยเน้นการทำงานที่ง่ายเร็ว ตัดพาก business logic ต่างๆไปอยู่ที่ module หางนอกแทน ฉะนั้นในการโอนข้อมูลเข้ามาเพื่อเขียนลงฐานข้อมูลก็จะเร็วมากและใช้พื้นที่น้อยกว่าอีกด้วย

InfluxDB ก็มีทั้งส่วนที่เป็น Commercial และ Opensource ซึ่งข้อแตกต่างหลักๆของ version ที่เสียเงินกับ Opensource นั้นก็จะเป็นในเรื่องของการ support และก็เรื่องของ High Availability หรือการ Scale out ระบบเพื่อรับรองการทำ 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 • Supported by InfluxData • 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 ด้วยกันคือ Telegraf, InfluxDB, Chronograf และ Kapacitor



โดยที่การทำงานก็จะเป็นอย่างรุปที่แสดงอยู่ด้านบนเลยครับ

- **Telegraf** จะเป็นตัวกลางในการรับข้อมูลจาก Source ต่างๆเข้ามาไม่ว่าจะเป็น Database อื่นๆหรือแม้กระทั่ง MQTT Message และรวมถึงจัดซื้อข้อมูลจาก InfluxDB และส่งกลับไปยังระบบอื่นด้วย ทำหน้าที่เหมือน ETL
- **InfluxDB** ก็เป็นหัวใจของ TICK Stack นี้เลยซึ่งก็คือ Time Series Database ของเรานั่นเอง เมื่อกำหนดชื่อฐานข้อมูลอย่างเดียว
- **Chronograf** อันนี้จะเป็น GUI ที่ใช้ในการ Manage InfluxDB รวมถึงการทำ Dashboard ด้วย (แต่เดี๋ยวเราจะไปใช้ Grafana ที่อ้อ)
- **Kapacitor** จะเป็นส่วนที่ทำในเรื่องของ Data Processing ต่างๆ เช่นเมื่อมีข้อมูลการตรวจสอบไฟฟ้าพักก็จะทำการ Call Center เกินเท่านี้ให้ทำการ Alert แจ้งเตือนไปยังหัวหน้างาน ซึ่งก็สามารถนำ Module การวิเคราะห์ดังๆเข้ามาใช้ได้ ไม่ว่าจะเป็นจากการตั้งค่าเอง หรือ Machine Learning ก็ได้

InfluxDB Setup

โดยการติดตั้งนี้จะเริ่มจาก InfluxDB กันก่อน โดยการเปิดหน้า SSH Command Windows ของ Google Cloud แล้วพิมพ์คำสั่งต่อตามค้างล่างนี้เลยครับโดยสเตปก็จะเป็น Download & Install → Start Service → Verify (ใครที่อ่านตรงนี้แล้วงง อาจต้องย้อนกลับไปอ่านที่ [ตอนแทรกร 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"
```

ด้าน InfluxDB ที่เราเพิ่งติดตั้งไปทำงานให้ออกมาดูก็ต้องແลี้ยวහລະກື ຈະແສດງຜລັບພົມທານບຣທັດລ່າງນີ້ແລຍກວັນ

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

Kapacitor Setup

ນາຄື່ງ Module ທີ່ສອງທີ່ຈະກຳໄຊໃຫຍ່ໃນການທຳ Data Processing ຈຶ່ງດ້າໄກໄມ່ໄດ້ໃຊ້ງານກີ່ສາມາຮອ່ານໄປກ່າຍເປົ້າຕິດຕັ້ງ

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
```

ຈຶ່ງດ້າຕິດຕັ້ງແລະ Service Start ເຮັນຮ້ອຍແລ້ວ ຜລັບພົມທີ່ໄດ້ຈະເປັນດັ່ງນີ້ກວັນ

ID	Type	Status	Executing Databases	Retention Policies

Telegraf Setup

Module Telegraf ນີ້ເຮັດວຽກໄດ້ວ່າເປັນອົກຫ້າໃຈຫລັກເລຍກື່ວ່າໄດ້ກວັນ ເພຣະຈະເປັນຕົວທີ່ເກີບຮາບຮາມ ດຶງຂໍ້ມູນຈາກ Source ດ່າງໆເພື່ອທີ່ຈະໄຫຍ້ເຂົ້າ InfluxDB ໃນຕົວຢ່າງນີ້ເຮັດວຽກເກີນ System Stat ຂອງ VM Instance ຂອງເຮົາກັນດ້າຍເພື່ອດູ້ load ຂອງ CPU / RAM ກັນ

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 โดยขั้นแรกให้เปิดไฟล์

```
/etc/telegraf/telegraf.conf
```

จากนั้นให้ดูในส่วนของ **Output Plugins** (จะใช้โปรแกรม vi, pico , nano อันนี้ก็แล้วแต่สะดวกกันเลยครับ)

```
[[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** นั้นก็ควรจะมีหน้าตาลักษณะแบบนี้ ในการดึงค่า **System Stat** ออกมายื่อไปเป็น **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

```

ชั่งจริงๆแล้วทั้งในส่วนของ Output/Input Plugins นั้นก็เป็นค่า Default ที่ติดตั้งมาพร้อมกับ Telegraf อยู่แล้วครับ เสร็จแล้วก็ทดลองรันคำสั่งนี้กันดู

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

ถ้า Telegraf ทำงาน ก็จะแสดงผลข่าวพื้นด้วยภาษา JSON เป็นรูปแบบ JSON ให้ได้เห็นกันเดี๋มหน้าจอเลย

Chronograf Setup

มาถึง Module สุดท้ายของ TICK Stack กันละครับสำหรับหน้า GUI เพื่อใช้ในการ Config และแสดงผล สำหรับผมแล้วเวลาไปทำบน Grafana น่าจะสะดวกกว่า แต่ไหนๆก็ใหญ่ๆแล้ว **Install** กันให้ครบเลยดีกว่ารับ 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

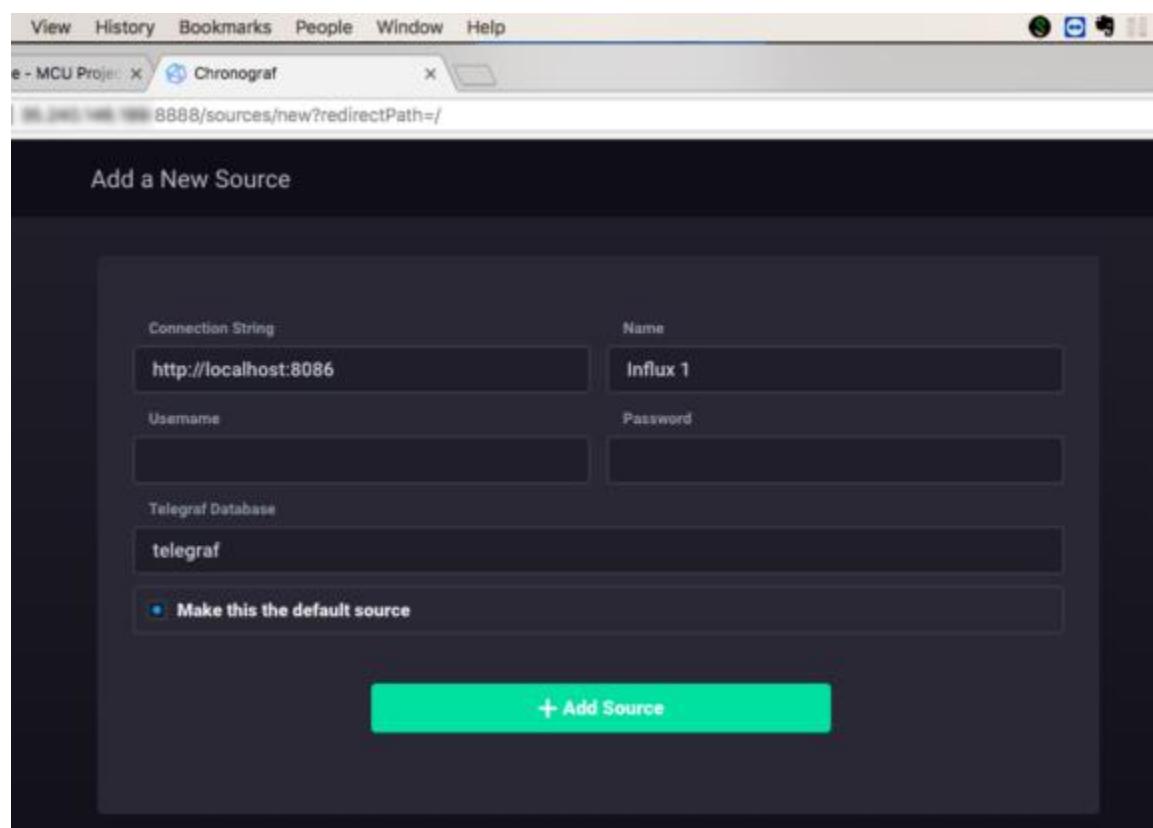
```
sudo systemctl start chronograf
```

Verify

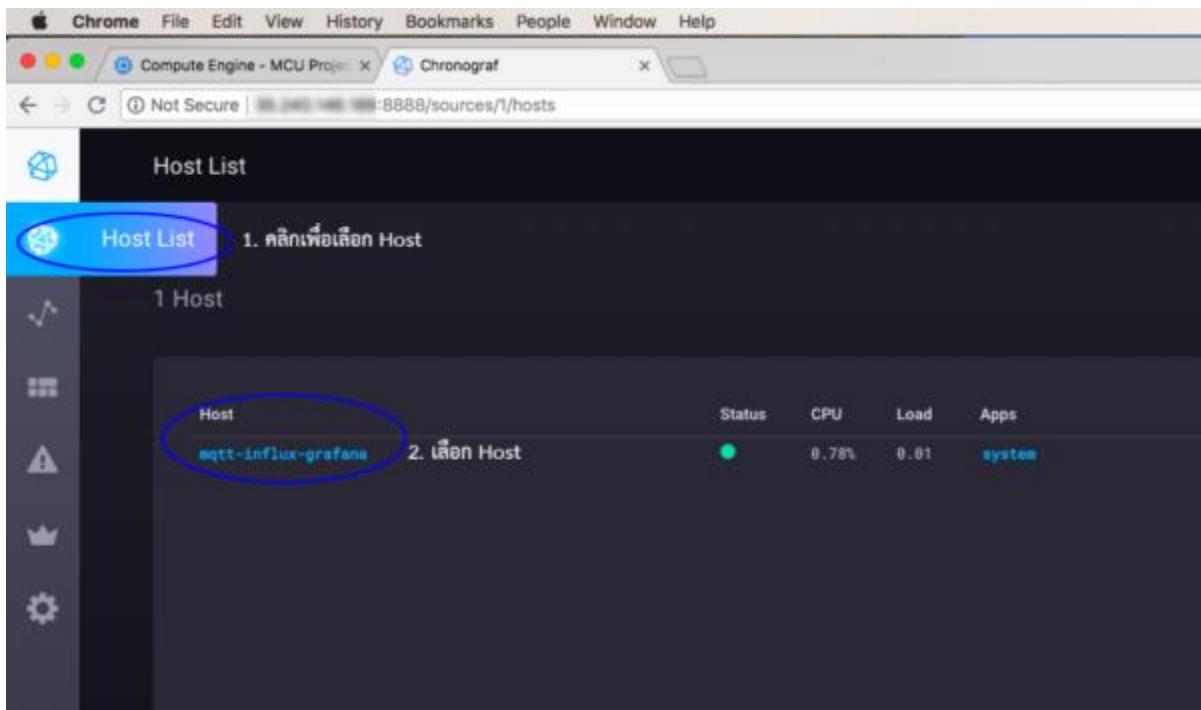
ขั้นตอนนี้ก็ให้เข้าไปที่

<http://localhost:8888>

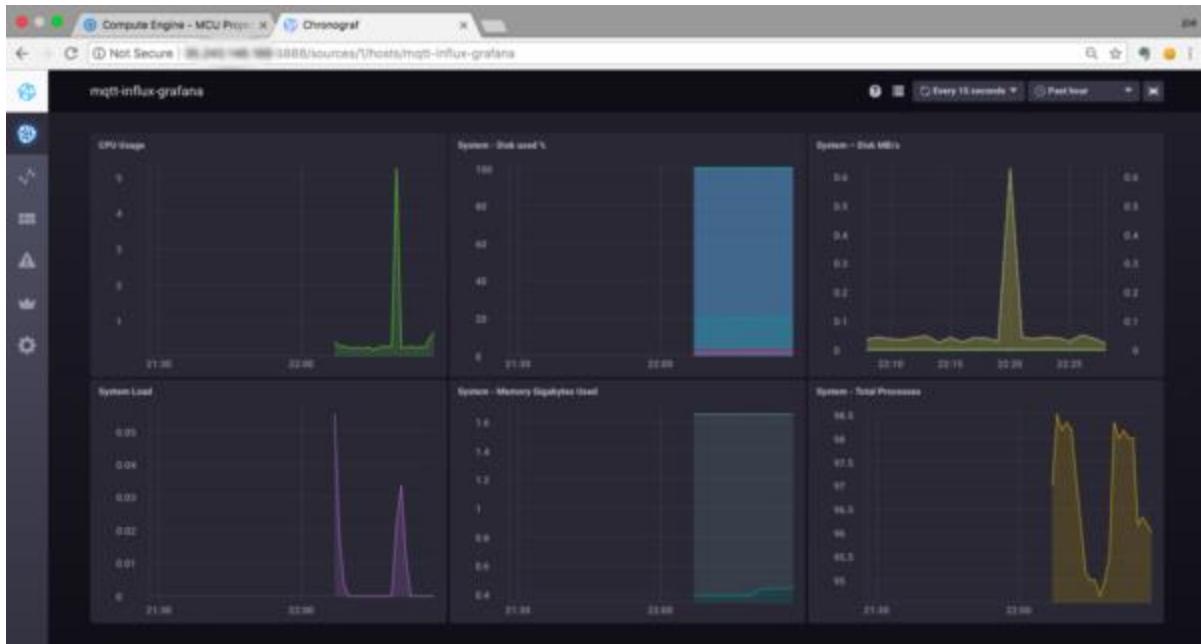
โดยเปลี่ยน local host เป็น ip หรือ hostname ของเราครับ ก็จะได้หน้า Web GUI ของ Chronograf ดังแสดงในรูป ด้านล่างนี้ สำหรับใครที่ใช้ Google Cloud ก็อย่าลืมเข้าไปเช็คในส่วนของ Firewall Rule ให้ Allow Port 8888 ด้วยนะครับ วิธีการก็หาได้จากตอนที่ 3.5 ได้เลย



หลังจากนั้นก็คลิกที่ปุ่ม +Add Source เลยครับ ค่าที่อยู่ให้คง Default ไว้เนื่องจากในการตั้งค่าเรียกนี้ Default Security ของ InfluxDB นั้นจะไม่มี Username/Password ซึ่งเดียวเราจะได้มา config กันในช่วงต่อไปครับ ซึ่งหลังจาก Add Source แล้วก็จะเข้าสู่หน้าอัดไป ให้ทำการเลือก Host ที่เราใช้ Plugin ของ Telegraf ในการดึง System Stat มาใส่ไว้ใน InfluxDB



ถ้าทุกอย่างทำงานได้อย่างถูกต้องเราก็จะได้หน้า Dashboard ที่ใช้ในการแสดงผล System Stat ที่ได้จากการลีจิ้งข้อมูลผ่านทาง Plugin Module ของ Telegraf และโอนข้อมูลที่ได้รับมาไว้ใน InfluxDB และนำมาแสดงผลบน Chronograf



แต่ยังไม่จบครับ เพราะ TICK Stack นั้นมีอยู่ด้วยกัน 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 แต่perm จะขอหยุดไว้เท่านี้ก่อนดีกว่า เพราะถ้าเราทำใน Grafana นั้นใน

ส่วนของ Dashboard และ Alert เราสามารถทำได้จาก Grafana ในที่เดียวกันเลย เราจึงมักเห็นการจับคู่กันเฉพาะในส่วนของ MQTT → Telgraf→ InfluxDB→Grafana หรือว่าเป็นสูตรสำเร็จสำหรับการเก็บข้อมูลมาแสดงผลของงาน IOT เลยครับ

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

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

ในส่วนของ admin_passwd ก็เปลี่ยนเป็นตามใจชอบได้เลยครับ

หลังจากนั้นก็ใช้คำสั่ง show users ในการเรียกดู User ทั้งหมดที่มีอยู่ ตามรูปด้านล่างนี้

```
mqtt-influx-grafana: /etc/influxdb$ influx
Secure | https://ssh.cloud.google.com/projects/mcu...
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 ที่แก้ไขก็จะเหมือนกับด้านล่างนี้

```
[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 ถ้าทดลอง list user ในระบบด้วยคำสั่ง 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
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

```
mqtt-influx-grafana: /etc/influxdb
Secure | https://ssh.cloud.google.com/projects/mcu-office-home...
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
```

ในส่วนของ [[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 ที่ต้องการดึง
```

มาตรฐานในส่วนของ influx line format กันบ้าง ซึ่งรูปแบบของข้อมูลที่ Telegraf รองรับนั้นก็มีหลากหลายรูปถึง JSON ด้วย ถ้าสนใจจะใช้ format อื่นก็จุใจกิจลิ่งค่านี้ได้เลยครับ [telegraf input data formats](#)

เนื่องจากตอนนี้เราจะมีการเขียนข้อมูลลงฐานข้อมูลของ InfluxDB 2 ตัวด้วยกัน โดยตัวแรกก่อนหน้านี้คือ Default ที่มาจากการติดตั้ง Telegraf ซึ่งเป็นข้อมูลพาก System Stat อีกอันก็จะเป็นข้อมูลที่เราได้มานา粗 Node ที่ส่งข้อมูลมาที่ MQTT Server (ในที่นี่จำลองส่งข้อมูลอุณหภูมิเท่านั้น) จะนั้นเราต้องบอก Telegraf ว่าข้อมูลไหนจะเก็บลงฐานข้อมูลไหน โดยการใช้ namedrop, namepass ตามด้าวอย่างด้านล่างนี้ครับ

```
[[outputs.influxdb]] #<-- ใช้เขียนข้อมูล System Stat ลง InfluxDB โดยเก็บที่นั่นข้อมูล 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]] #<-- ใช้พิมพ์ชื่อคุณ System Stat ลง InfluxDB ได้ยกเว้นที่มีชื่อคุณ 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 เลยก็วัน

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

ชี้การเก็บข้อมูลใน InfluxDB จะแยกเป็น Measurement, TAG Key – TAG Value และ Field Key – Field Value จากตัวอย่างข้อมูลข้างบนนี้ ผ่านหน้า command line แล้วใช้คำสั่งแสดงค่าดังต่อไปนี้

```

Secure | https://ssh.cloud.google.com/projects/mcu-office-home
instances/mqtt-influx-grafana?authuser=0&hl=en
cybercnode@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 แต่บางครั้งนั้นอาจจะไม่ได้มีข้อมูล repository ของ Grafana อยู่ด้วยจะนะนั้นเราต้องทำการเพิ่มรายการของ Grafana เข้าไปในลิสต์ก่อนที่จะทำการติดตั้ง

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

จากนั้นตามด้วยคำสั่ง

```
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 ทำงานเป็นปกติก็จะขึ้นข้อมูลแสดงสถานะดังนี้ครับ

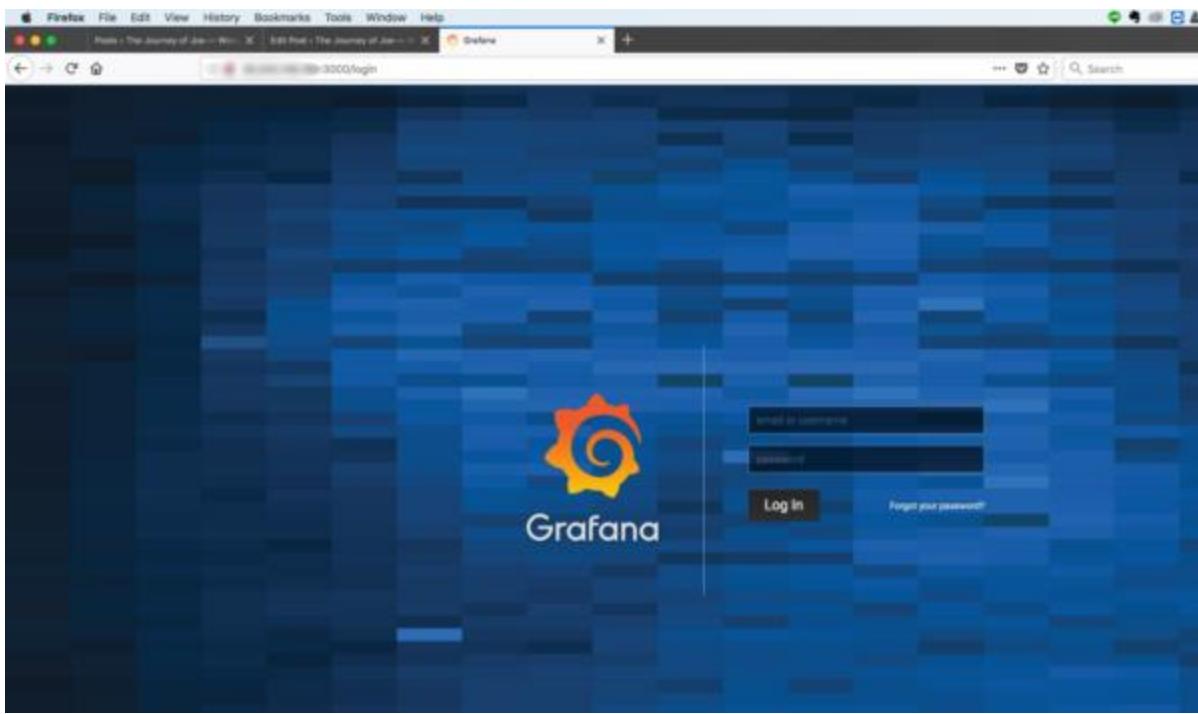
```
● Secure | https://ssh.cloud.google.com/projects/mcu-office-home
  mqtt-influx-grafana:/var/log/mosquitto
  x-grafana?authuser=1&hl=en_GB&proj...
  grafana-server.service - Grafana instance
   loaded: loaded (/usr/lib/systemd/system/grafana-server.service; disabled; vendor preset: enabled)
   Active: 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)
```

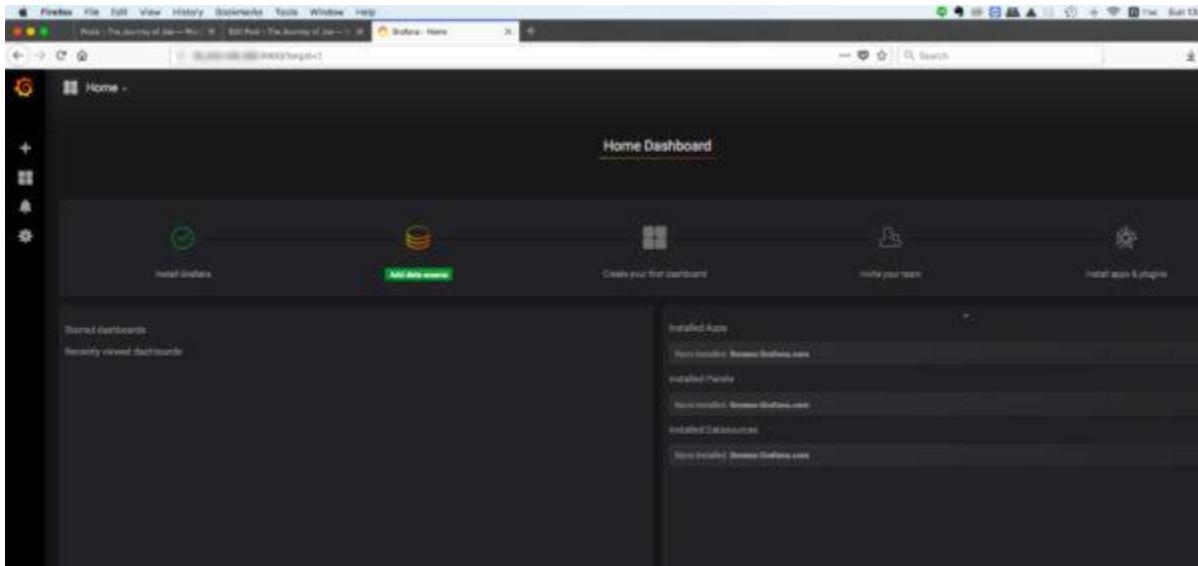
จากนั้นกดตั้งค่าให้ Grafana เริ่มต้นทุกครั้งที่มีการรีบูตด้วยคำสั่ง

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

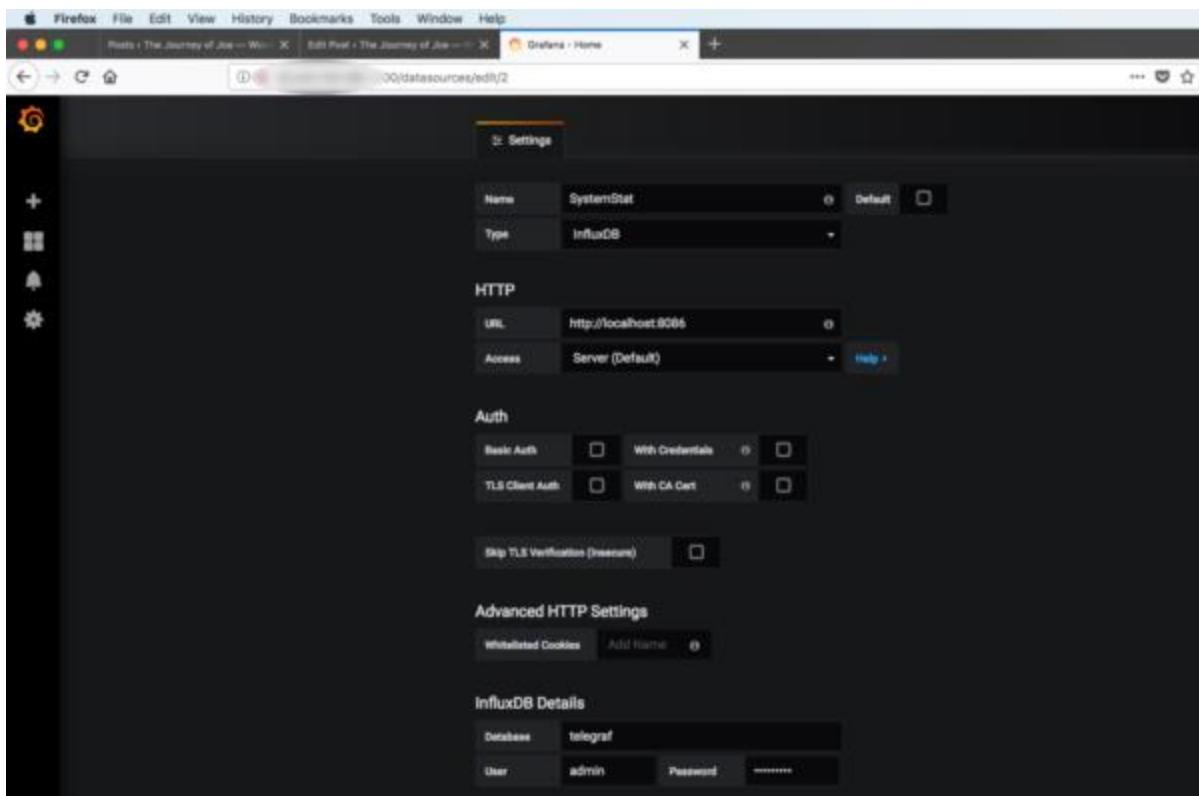
ช่องค่า Default Port ของ Grafana นั้นคือ port 3000 ดังนั้นถ้าใช้ Google Cloud ก็อย่าลืมไป allow port ให้ connection จากทั่วโลกสามารถเข้าถึง VM Instance ของเราราจาก port 3000 ได้ด้วยนะครับ ช่องการใช้งานก็เข้าได้จากหน้า Browser เลย



โดยที่ Default Username/Password ตั้งค่าเป็น admin/admin ครับ เมื่อ login เข้าไปแล้วระบบจะให้เราเปลี่ยน password ของ admin ใหม่ทันทีก็จะเข้าสู่หน้าจอต่อไป ซึ่ง Grafana นั้นก็ออกแบบขั้นตอนเป็นสเตปมาให้อย่างเดียวติดตั้งเสร็จก็เพิ่ม Datasource จากนั้นก็สร้าง Dashboard ครับจะใช้คนเดียวก็ไม่ได้ ก็สามารถ Invite คนอื่นเข้ามาใช้งานร่วมกันได้ รวมถึงรองรับ Plugin ต่างๆด้วย



เราจะมา Add Data source แรกกัน ซึ่งก็คือ Data source System Stat ที่ได้มาจากการดึงข้อมูลของ Telegraf ในตอนด้านบนนั่นเอง ให้คลิก Add Data Source จากนั้นก็ใส่ข้อมูลตามที่เรา config ไว้



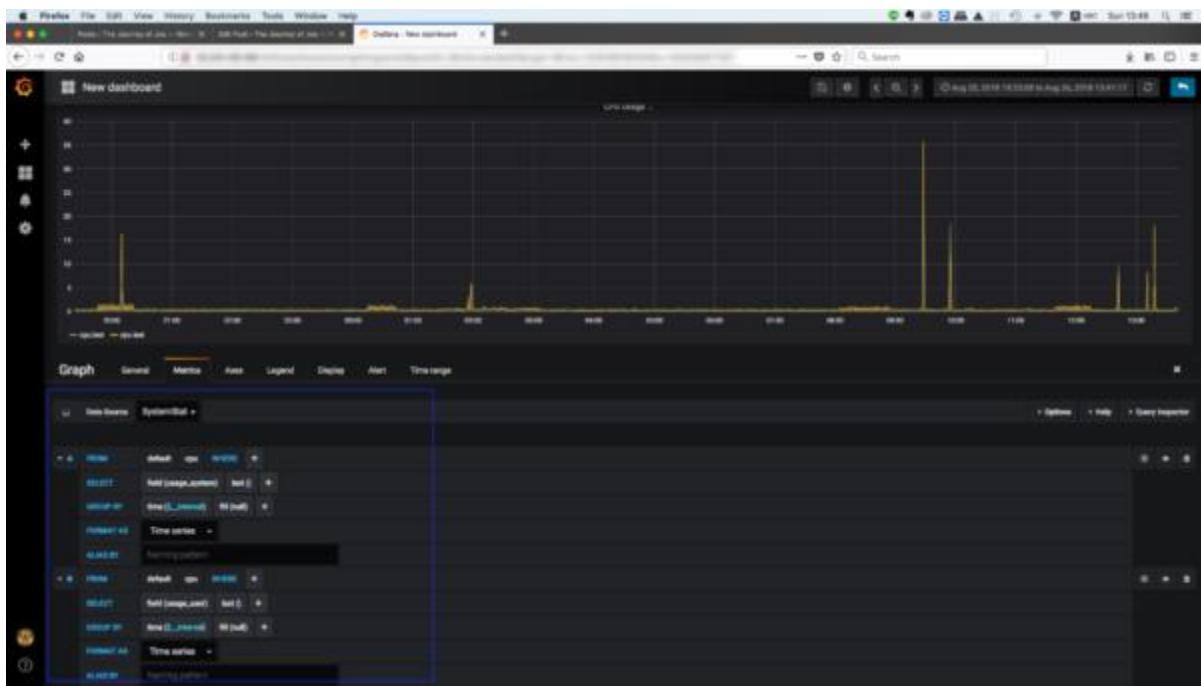
หลังจากที่เราเพิ่ม Data Source แล้วเราไปแล้วขั้นตอนที่ไปก็เป็นการสร้าง Dashboard ของเรากัน โดยเข้าไปที่ New Dashboard และเพิ่ม Graph เข้าไปยัง Dashboard แรกของเรา



จากนั้นคลิกที่ชื่อ Panel แล้วเลือก Edit เพื่อทำการเลือก Data Source ว่าจะเอาข้อมูลไหนมาสร้างกราฟ



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



หลังจากนี้ก็เข้าอยู่กับการประยุกต์ใช้งานกันละครับ ถ้าได้อ่านจากตอนที่ 1 รวมตอนแรกจะมีตอนที่ 5 ที่ปิดจบ Series ของการใช้งาน Painlessmesh แล้ว ก็จะทำได้ตั้งแต่การใช้งาน nodemcu ในการรับและส่งข้อมูลผ่านทาง mesh network การ bridge ข้อมูลผ่าน serial communication, lora network แล้วต่อไปยัง MQTT broker อย่าง Mosquitto MQTT ซึ่งทำให้เราสามารถควบคุมอุปกรณ์ผ่านทาง MQTT หรือนำข้อมูลไปเก็บใน Time Series Database อย่าง InfluxDB และนำไปแสดงผลผ่านทาง Dashboard ในตอนที่ 5 นี้

Arm-Pelion Full Stack IoT Platform

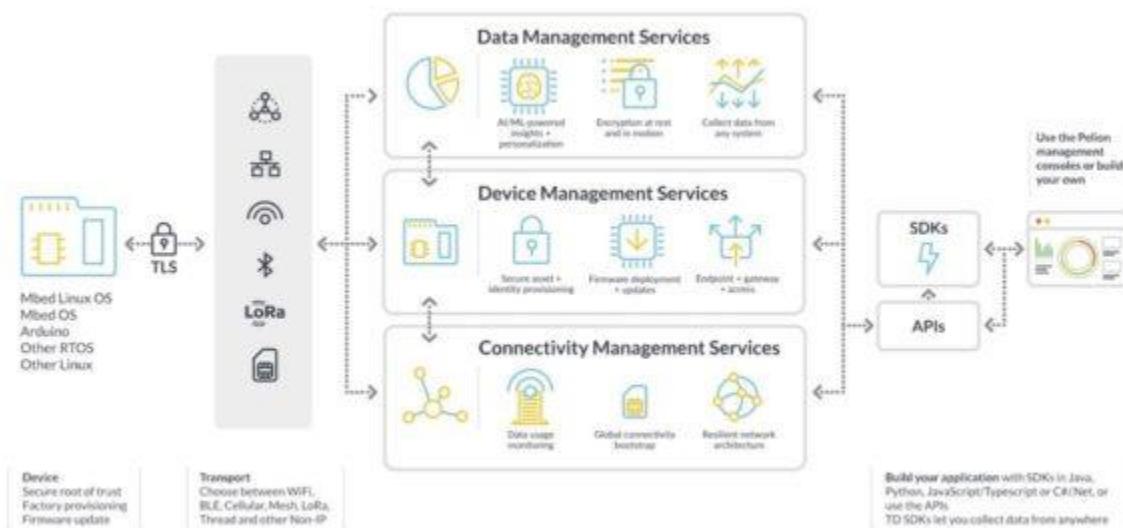
<https://www.techtalkthai.com/arm-pelion-full-stack-iot-platform/>

รู้จัก **ARM Pelion** แพลตฟอร์ม IoT จาก ARM จัดการทุกอย่างครบจบในที่เดียว

เมื่อวันที่ 30 กรกฎาคมที่ผ่านมาทาง ARM ได้จัด IoT Workshop ขึ้นเพื่อนำเสนอแนวคิดต่างๆของการนำเทคโนโลยี IoT ไปใช้ในธุรกิจ ที่มีงาน TechTalkThai ได้เข้าไปร่วมงาน และทำความรู้จักกับ ARM Pelion IoT Platform จึงขอจะมาเล่าให้ฟังกันคร่าวๆว่าเจ้าแพลตฟอร์ม IoT นี้ มีความสามารถอย่างไร และเหมาะสมกับธุรกิจแบบไหนบ้าง

IoT นั้นเป็นหนึ่งในเทคโนโลยีที่หลากหลายที่สุดให้เป็นมาตราศรีในปี 2019 หากความสามารถในการรวมรวมข้อมูล ซึ่งนับว่าเป็นวัตถุคุณภาพในการทำธุรกิจที่ขาดไม่ได้เลยในยุคปัจจุบัน จนถึงตอนนี้ หลากหลายองค์กรต่างเริ่มศึกษา IoT กันบ้างแล้ว แต่ความท้าทายใหม่ที่ธุรกิจมักจะเผชิญหน้าก็คือการจัดการและส่งกระบวนการ IoT ให้ใช้งานเก็บข้อมูล ได้จริงดีมีประสิทธิภาพ ปลอดภัย และมีระบบจัดการที่ดี ดังนั้นจึงมีการพัฒนาแพลตฟอร์ม IoT ขึ้นเพื่อช่วยธุรกิจในการแก้ไขปัญหานี้

ARM Pelion IoT Platform ก็เป็นหนึ่งในแพลตฟอร์ม IoT ที่จะเข้ามาช่วยลดความซับซ้อนของการนำ IoT ไปใช้งานในธุรกิจ แพลตฟอร์ม Pelion นี้แบ่งออกเป็น 3 ส่วน ตามการใช้งาน คือ Connectivity Management Services, Device Management Services, และ Data Management Services โดยทั้ง 3 ส่วนจะทำงานร่วมกันภายใต้ระบบรักษาความปลอดภัย ซึ่งเป็นหลักสำคัญที่สุดในการพัฒนาผลิตภัณฑ์ทุกๆ ด้านของ Arm



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

Pelion จะช่วยให้ธุรกิจจัดการกับเครือข่าย อุปกรณ์ในเครือข่าย และข้อมูลที่เก็บมาได้ง่ายขึ้น โดยสามารถทำงานร่วมกับอุปกรณ์ ระบบเครือข่าย คลาวด์ และข้อมูลได้หลากหลายรูปแบบ อีกทั้งยังมีความปลอดภัย และสามารถช่วยในการนำข้อมูลไปวิเคราะห์และแสดงผลเบื้องต้นได้ด้วย

ผู้จัดแพลตฟอร์มนี้ไปคร่ำแคล้ว่องมาจะเลือกกันว่าส่วนประกอบทั้ง 3 ส่วน อันได้แก่ Connectivity Management, Device Management, และ Data Management นั้นประกอบไปด้วยอะไร และมีจุดเด่นอย่างไร

Connectivity Management

การเชื่อมต่อในเครือข่าย IoT นั้นมีอยู่หลายรูปแบบ และมีรายละเอียดปลีกย่อยที่ธุรกิจจะต้องจัดการอยู่เสมอ Pelion จะช่วยให้องค์กรสามารถจัดการการเชื่อมต่อได้อย่างมีประสิทธิภาพ ปลอดภัย และพร้อมต่อการสเกลเครือข่ายที่นี่ไปถึงระดับโลก โดย Connectivity Management ของ Pelion มีความสามารถที่น่าสนใจ ดังนี้

Global Cellular

Pelion จะช่วยให้อุปกรณ์ IoT สามารถเชื่อมต่อผ่านเครือข่ายได้ไม่ว่าอุปกรณ์นั้นจะอยู่ที่ใดในโลก ผ่านเวนเดอร์เพียงเจ้าเดียว โดยกลไกของ Pelion จะช่วยเชื่อมต่อสัญญาณจากชิมของอุปกรณ์ไปยังเครือข่ายท้องถิ่นที่ Pelion ได้ทำขึ้นตอกลังไว้ ลดการระความปวดหัวในการติดต่อกับผู้ให้บริการเครือข่ายในแต่ละประเทศ

Protocol เชื่อมต่อทั้ง IP และ Non-IP

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

ใช้ได้ทั้ง eSIM และชิมแบบปกติ

ธุรกิจสามารถสั่งผลิตอุปกรณ์ที่มีระบบ eSIM ผ่าน ARM ได้ตามต้องการ โดย eSIM ที่ได้มา กับอุปกรณ์นั้นจะรองรับการเชื่อมต่อกับเครือข่ายกว่า 600 เครือข่ายทั่วโลก และหากต้องการเปลี่ยนเครือข่าย ทีมงานต้องตั้งค่าใหม่ให้กับหิ้ง แล้วในส่วนของชิมแบบปกติของ Pelion ก็ให้บริการซึ่งการคัดในทุกขนาด อีกทั้งยังมีแผนที่จะพัฒนาไปจนถึง iSIM ที่มีขนาดเล็กกว่า eSIM มากด้วย

Network Infrastructure

Pelion ได้พัฒนาโครงสร้างพื้นฐานของเครือข่ายให้สามารถทำงานร่วมกับผู้ให้บริการเครือข่ายทั่วโลก ได้อย่างมีประสิทธิภาพสูงสุด โดยมีทั้งความเสถียร ขีดหยุ่น และเป็นไปตามกฎข้อบังคับด้านข้อมูลของแต่ละประเทศ ในการใช้ Pelion ผู้ใช้งานสามารถเลือกได้ว่าจะส่งข้อมูลจากอุปกรณ์ไปยังแอปพลิเคชันผ่านทางโอดีไอ เช่น IPSEC, Open VPN, ผู้ให้บริการ Cloud, หรือทางชื่อมที่ธุรกิจเข้ามาใช้โดยเฉพาะ (Leased Line)

Device Management

Device Management ของ Pelion นั้นจะช่วยให้องค์กรสามารถจัดการกับอุปกรณ์และการเชื่อมต่อกับอุปกรณ์ผ่านช่องด้วยได้โดยสะดวก ไม่ว่าจะเป็นการเขียนซอฟต์แวร์แบบ Embedded หรือว่าการเขียนแอปพลิเคชันด้านบนอย่าง Web App ที่สามารถ



(ภาพ: ARM)

โดยภายในโซลูชัน Device Management ก็จะมีโมดูลในการจัดการเรื่องต่างๆ ให้อ่ายกว้างขึ้น ตั้งแต่เรื่องการอัพเดต Firmware ซึ่งไม่ง่ายหากไม่มี อุปกรณ์ที่หลากหลายและมีจำนวนที่มากในเครือข่าย, Access Management ซึ่งจะร่วมยกระดับการเข้าถึงอุปกรณ์และการควบคุมแต่ละส่วน, Connector ซึ่งจัดการการเชื่อมต่อกับอุปกรณ์หลากหลายประเภท การออก Certificate เข้าใช้ระบบ การเข้าทะเบียนอุปกรณ์แล้วล็อค และการเก็บสถิติ, Device Directory ซึ่งช่วยในการแบ่งกลุ่ม ค้นหา เรียกดู และเช็คสถานะของอุปกรณ์แต่ละตัว, ไปจนถึงการรักษาความปลอดภัยในการส่งต่อรหัสผ่านเครือข่าย Wifi โดยทั้งหมดนี้จะดำเนินการตามชั้นการ Lifecycle ของอุปกรณ์ทั้งหมด ตั้งแต่การ Onboard เข้าระบบ ไปจนถึงการใช้งานและบำรุงรักษา

Device Management นั้นสามารถพูดคุยกับอุปกรณ์ IoT ผ่านโปรโตคอลหลากหลาย โดยเดaf พา LwM2M ซึ่งช่วยให้แนบชิดกับอุปกรณ์ที่ต้องการให้ต่อเนื่องโดยที่มีลักษณะคล้ายๆ REST Model และ CoAP ซึ่งช่วยประหยัดแบนด์เดรอะของอุปกรณ์ได้มากกว่า HTTP ราว 8-10 เท่า และในการเชื่อมต่อ กับแอปพลิเคชันซึ่งเป็นปลายทางอีกด้านหนึ่ง Pelion ได้เตรียม REST API และ SDK ในภาษา Java, Python, JavaScript และ .NET ไว้ให้พัฒนาแอปพลิเคชันกันได้โดยง่าย

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

Data Management

เนื้อประสึ่งค์หลักของการจัดตั้งระบบ IoT นั้นคือการสร้างระบบจัดเก็บข้อมูลที่จะช่วยให้องค์กรสามารถเรียกข้อมูลเหล่านั้นขึ้นมาวิเคราะห์ที่เป็นความรู้ที่มีประโยชน์ต่อธุรกิจได้ แนะนำว่า Pelion ยังไม่ลืมความสำคัญของส่วนนี้ จึงได้พัฒนาระบบจัดการข้อมูลครบวงจรที่จะช่วยตั้งแต่การจัดเก็บ นำข้อมูลมาใช้ตัดสินใจแบบ Real-time และรักษาความปลอดภัยและความเป็นล่าวนห้องข้อมูล โดยมีกลไกรองรับการสแกลเพิ่มที่ ทำให้อ่องค์กรไม่ต้องกังวลว่าระบบจะทำงานได้เมื่อเวลาผ่านไป อุปกรณ์ในเครือข่าย IoT เพิ่มมากขึ้นเมื่อเวลาผ่านไป

ใช้สูตรหลักของส่วนนี้ กือ **ARM Treasure Data** ซึ่งเป็นซอฟต์แวร์จัดการและวิเคราะห์ข้อมูลที่เชื่อมต่อมาจาก Pelion ได้จบครบในตัวเดียว โดยมีเครื่องมือต่างๆพร้อมให้เลือกใช้งาน เช่น ระบบ Predictive Analytics การสร้าง Customer View 360 องศาจากข้อมูลการใช้งาน การวิเคราะห์ข้อมูลเพื่อ Cross-sell และ Upsell และการสร้างระบบ Recommendation เป็นต้น ซึ่ง ใช้สูตรนี้หลายอย่างคือที่ได้นำไปใช้งานเพิ่มประสิทธิภาพให้กับการทำงานในอุตสาหกรรมมากมาย เช่น อุตสาหกรรมถ้าปลีก อุตสาหกรรมพลังงาน อุตสาหกรรมการผลิต และอุตสาหกรรมที่น่าอึ้งมาก

Grafana Dashboard

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

Grafana กือ open source Dashboard tool เรียกง่าย ๆ กือเครื่องมือในการสร้าง Dashboard ฟรี นั่นเอง

โดย Grafana จะทำงานร่วมกับ Datasource ต่าง ๆ เช่น Graphite, InfluxDB, OpenTSDB หรือ Elasticsearch ฯลฯ ช่วยให้ users สามารถสร้างและแก้ไข Dashboard ได้อย่างง่ายๆ ครอบคลุมรูปแบบกราฟหลายประเภท

จุดเด่นของ Grafana

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

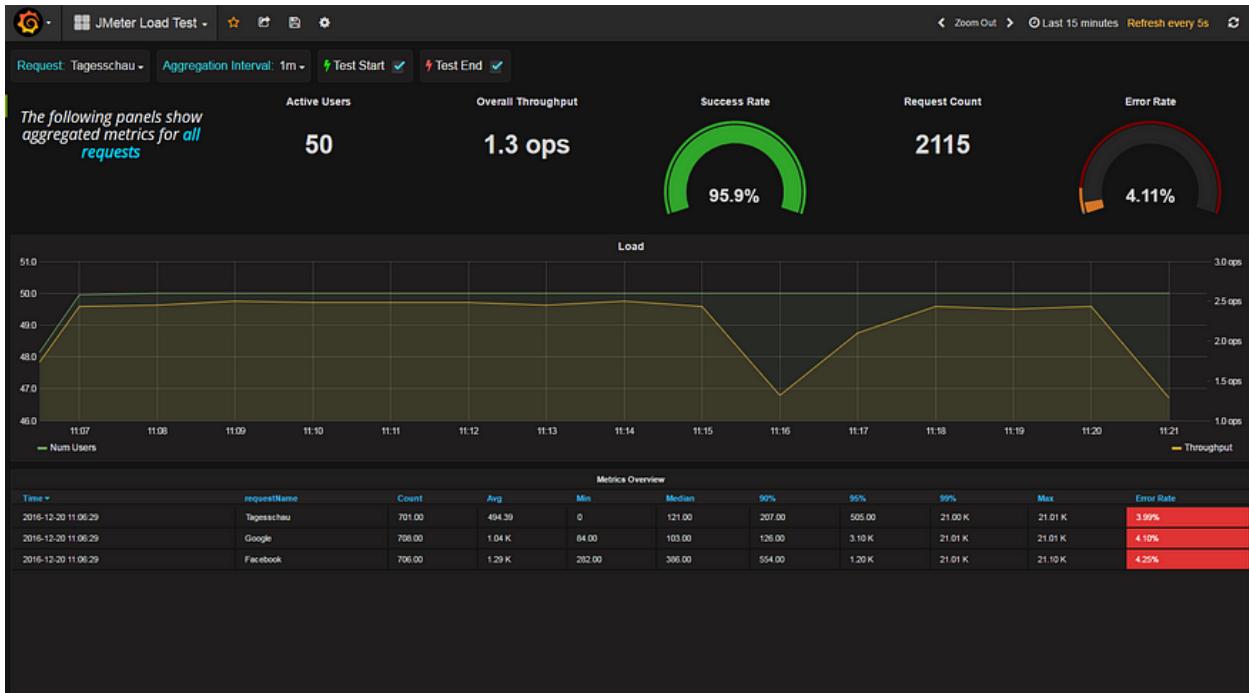
ตัวอย่างการใช้งาน Grafana Dashboard

- Monitoring Server ใช้งานร่วมกับ Influxdb และ Telegraf



credit : <https://grafana.com/dashboards/1443>

- Monitoring Realtime result สำหรับ Jmeter ใน non-gui mode



credit : <https://grafana.com/dashboards/1152>

การติดตั้ง Grafana Dashboard

- ดาวน์โหลด [ที่นี่](#)

สำหรับ Windows (x64)

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

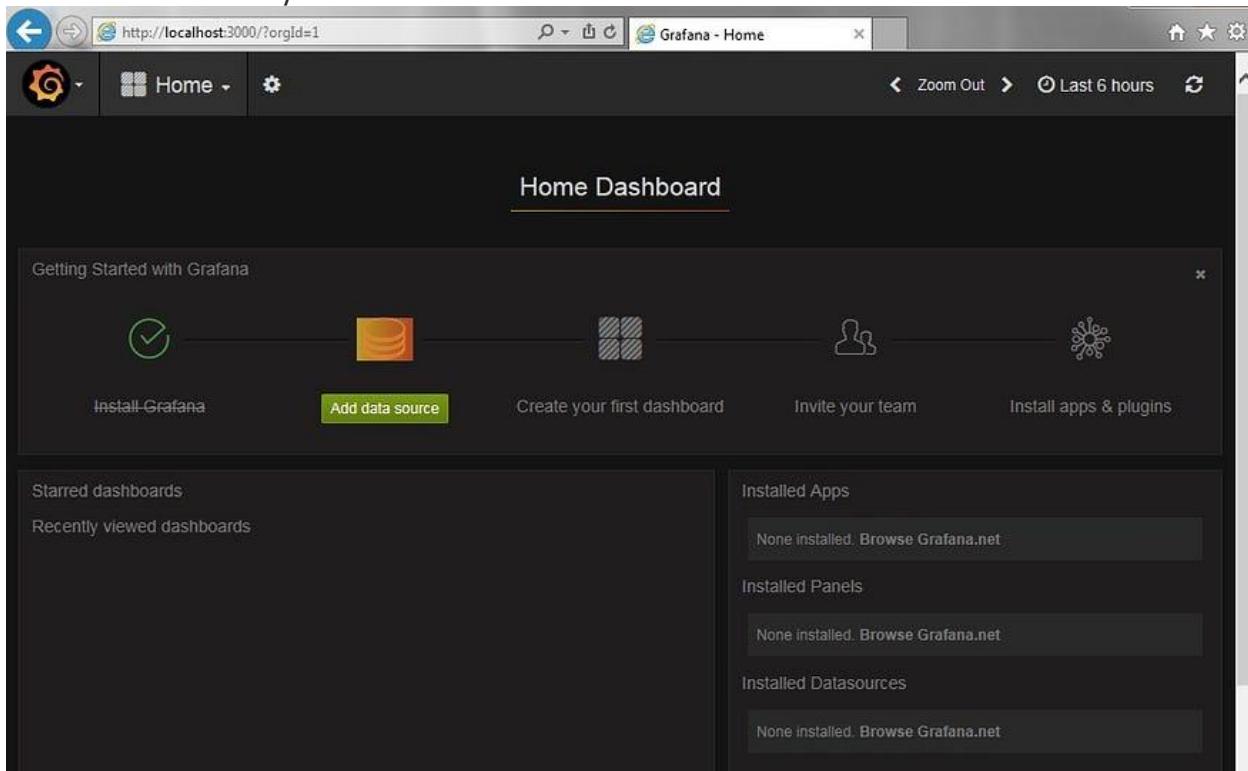
```
C:\>grafana-4.4.1>bin\grafana-server.exe --config conf\custom.ini
*[32mINFO*[0m[07-28:22:33:40] Starting Grafana *[32mlogg
er+[0m=main +[32mversion+[0m=4.4.1 +[32mcommit+[0m=6a9f8caa4 +[32mcompiled+[0m=2
017-07-05T14:15:04+0700
*[32mINFO*[0m[07-28:22:33:40] Config loaded from *[32mlogg
er+[0m=settings +[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=sqllite +[32mdbtype+[0m=sqllite3
*[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
*[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
```

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

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



เท่านี้ก็สามารถรีเมต์ตันใช้งาน Grafana Dashboard ได้แล้ว ครั้งหน้าจะมาแนะนำการใช้งานร่วมกับ Influxdb ในการ Monitoring Server และ Monitoring realtime jmeter