

Week0510.20230502-Grafana and InfluxDB

รหัส B6310158 ชื่อ-สกุล ณัฐพงศ์ โต๊ะแอก

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/>

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

จัวหัวตอนที่ 1 แล้ว แสดงว่าต้องมีมากกว่า 1 ตอนแน่ แต่ไม่ต้องห่วงว่าจะนาน มีคนปรามาสไว้ว่าให้เวลาตั้ง 6 เดือนแน่น และคงทำไม่ได้ ป้าดดดด ไหนๆก็ไหนๆแระ เขียนจะเลยเรื่อง Mesh Network ซึ่งก็ไม่ใช่เรื่องใหม่มอะไร ซึ่งเรื่อง Mesh Network ที่จะเขียนนี้ก็จะแบ่งออกเป็น 4 ตอนด้วยกัน ไหนๆเขียนแล้วก็เล่นใหญ่ไปเลย นะอ้อเจ้า บทความนี้เขียนเอามันส์ล่วนๆ ถือว่าทบทวนความรู้กัน ไม่ตรม่า มาม่า ไดๆทั้งสิ้น คนอ่านควรมีความรู้การใช้งาน Arduino IDE และบอร์ด MCU โดยเฉพาะ ESP8266 และ ESP32 นาบ้าง

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

Introduction

อะมาดูนิยามกันจะหน่อยเด้าบวกว่า Mesh Network เนี่ยมันก็คือโครงสร้างของ Network แบบหนึ่ง แหล่งที่มีลักษณะเป็น Node ซึ่งเชื่อมต่อถึงกันหมวดและทำงานร่วมกันในการส่งผ่านข้อมูลจาก node นึงไปยังอีก node นึง ซึ่งการทำงานในการส่งผ่านข้อมูลตรงนี้แหละที่ไม่ขึ้นตรงต่อกันแบบตายตัว นั่นหมายความว่าโครงสร้างของ mesh นั้นสามารถปรับเปลี่ยนได้ตลอดเวลา คุยกันตลอด เช่นถ้าจะส่งข้อมูลจากจุด A ไปจุด B จะไปเส้นทางไหน ถ้าเส้นทางจาก A->B->C->D แล้วถ้า Node B ล่ม จะ route ข้อมูลไปที่ Node ไหนต่อเพื่อให้ถึงปลายทาง (Self-Organize / Self-Configure) ซึ่งมันจะดีมากๆเลยในการติดตั้งๆ อยู่อุปกรณ์เข้าไปใน Mesh และมันก็คุยกันเองได้ เจ๊งประหลาด

ซึ่ง Concept ของ Mesh Network คร่าวๆก็อย่างที่อธิบายไปข้างต้นในเนตมีให้อ่านเพียบทั้งไทยและเทศ ซึ่งที่ตอนนี้เรายังไงลั้วเราภักดีมาก ก็จะเป็นอุปกรณ์ประเภท Home Automation ซึ่งเริ่มเข้ามาทำการตลาดในบ้านเรายะชั้น ในตลาดก็จะมีมาตรฐานของการสื่อสารผ่านอุปกรณ์พวกนี้อยู่ 2 ค่ายใหญ่นั้นก็คือ Zigbee และ Z-Wave โดยคุณสมบัติของทั้งสองค่ายนี้เลยก็คือ Mesh Network และ Low power Consumption ฉะนั้นในการติดตั้งอุปกรณ์ไม่ว่าจะ Zigbee หรือ Z-Wave เข้ากับ Gateway นั้น สามารถทำได้ง่ายมาก ไม่ต้อง config routing หรือเดินสายอะไรให้ยุ่งยาก แต่ยังขยายพื้นที่การใช้งานออกไปได้ไกลขึ้นตาม node ปลายทางที่เชื่อมถึงกันด้วย

คราวนี้เรามาดูผู้ของตัว micro controller กันบ้างซึ่งในตลาดก็มีหลายค่าย ถ้าใครอยากได้ mcu ที่มี z-wave ในตัว ใช้ Arduino IDE พัฒนาและรองรับการทำงานแบบ Mesh Network ของค่าย z-wave ได้ก็มี แต่วันนี้เราจะคุยกันถึงพระเอกตัวหลักของเรานั่นก็คือ ESP8266 กับ ESP32 จากค่าย Espressif

เจ้า ESP8266 / ESP32 ที่บางคนเรียกว่าเป็น Arduino Killer นี้มาพร้อมคุณสมบัติ WiFi ในตัวซึ่งหมายความว่าจะมาทำอุปกรณ์ประเภท IoT โดยที่สามารถทำตัวเองเป็นได้ทั้ง Client/Server ที่มี Built-in WebServer

ในตัว เป็น Access Point ก็ได้ สารพัดประโยชน์จะใช้งานจากการที่เป็น WiFi Enabled ซึ่งทาง Espressif ที่เป็นคนพัฒนา ESP8266/ESP32 ก็เลือกเห็นเหมือนที่หลายๆท่านเห็นแหล่วว่า Mesh Network มันเจ๋งนะ ก็เลยพัฒนา ESP-Mesh Protocol ขึ้นมา

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.

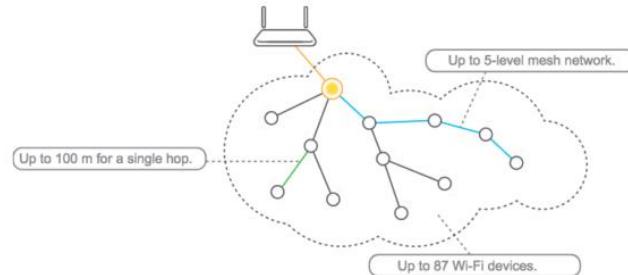
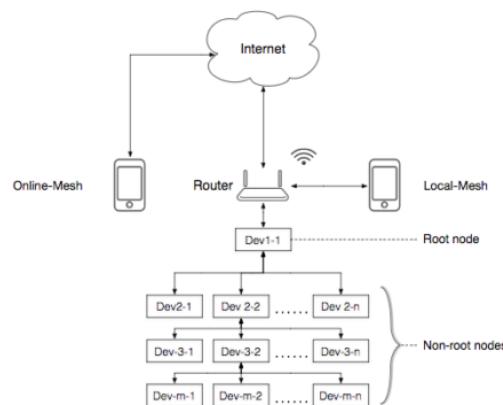


Figure 1-4 shows the mesh network diagram.



ซึ่งถ้าดูจากเอกสารของ ESP8266 แล้วเนี่ย สามารถที่จะเชื่อมต่ออุปกรณ์กันได้ถึง 87 ตัวเลยทีเดียวแล้ว ลองคิดดูว่าในแต่ละ hop ของ Node อย่ากว่าแต่ 100 เมตรเลย แค่ 50 เมตรก็พอ อุปกรณ์ 87 ตัวนี่ครอบคลุมพื้นที่ได้เยอะมากๆ พังดูดีอีกแล้ว แต่ถ้าไปดูในส่วนของ การใช้งาน ESP-Mesh บอกได้เลยว่ามีนีตี้บ รวมถึงจากข้อมูลที่หามามีคนเจอปัญหาค่อนข้างเยอะในการใช้งานโดยเฉพาะเรื่อง Self-Organize/Self-Configure ฉะนั้นเราจะข้าม ESP-Mesh Protocol กันไป

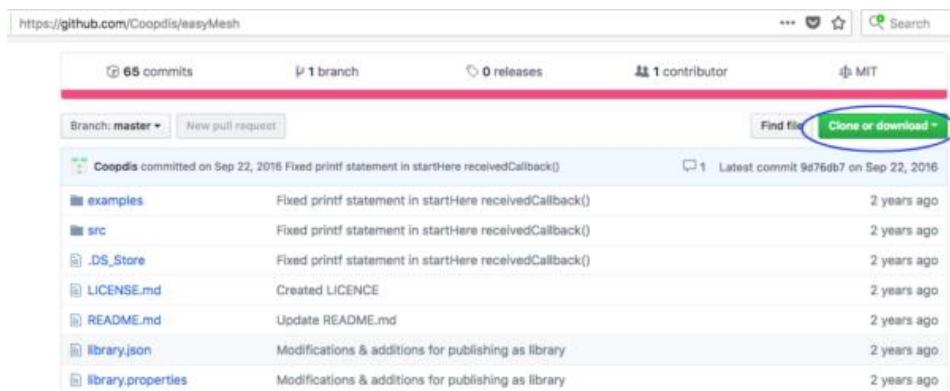
ใครอยากรอんเรื่อง ESP-Mesh Protocol อ่านได้จากที่ลิงค์นี้เลย “ESP-Mesh Protocol by Espressif”

จากการศึกษาค้นคว้าอย่างหนัก (จริงๆก็ไม่เท่าไหร่หรอก แค่ Google และ ก็อปปี้มาไว้เรื่อยๆแค่นั้นแหละ ยุคเดี๋ยวนี้กิจกรรม 2540 ที่ผมเรียน MCS51 อีก) เลยทำให้พบว่าก็มีคนเจอปัญหาจาก ESP-Mesh Protocol จริงๆและใช้งานค่อนข้างยาก อย่าไปเชือครับลองเปิด pdf ดู มีมีนอะครับ ก็เลยหาอ่านมาเรื่อยๆมาพบกับ library ที่ชื่อว่า EasyMesh กับ Painlessmesh อ่านแค่ชื่อก็รู้สึกใจซื้นขึ้นมาแล้วทั้งสองตัว ตัวนึงง่ายๆอีซี่ อิกตัวไม่ปวดหัว พอก่อนไปเรื่อยๆอิกเลยพบว่า Painlessmesh เนี่ยทางคุณ BlackEdder (อันนี้ไม่รู้นามสมมติหรือชื่อจริงนะ) พัฒนาต่อเนื่องมาจาก EasyMesh ซึ่งหมายความว่า Painlessmesh ใช้ได้ทั้ง ESP8266 และ ESP32 อิกด้วย

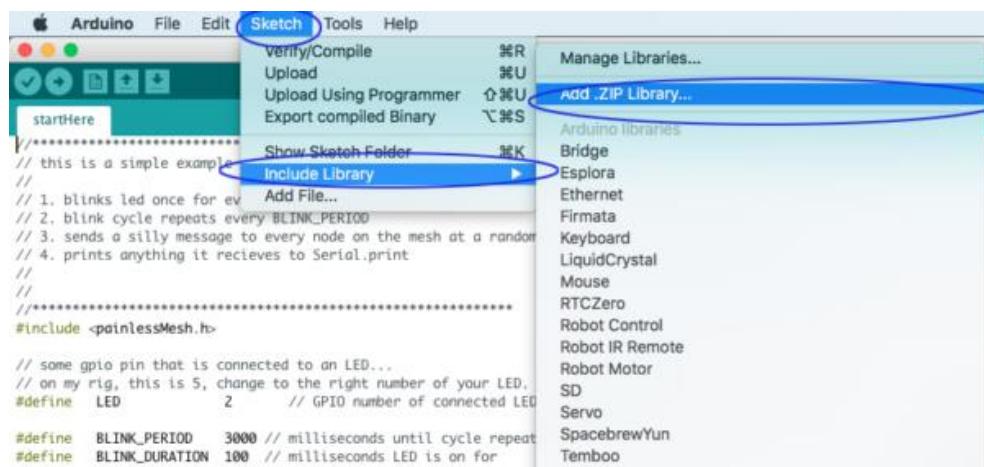
ว่าแล้วก็มาเริ่มกันเลยดีกว่า เกริ่น Intro จะเยอะเลย ไปค้นกระปองของเล่นที่พอดีต่อ กับเซนเซอร์อื่นๆ หรือทดลองงานอื่น ได้มา 17 ตัว เดียวเราจะเอา 17 ตัวนี้แหละมาลองทำ Mesh Network กัน ก็มี ESP8266 15 ตัว (NodeMCU 4 ตัว Wemos D1 6 ตัว ESP-01 อีก 5 ตัว) กับ ESP32 อีก 2 ตัว (WRoom32 กับ Heltec Lora)

เตรียมความพร้อม

ซึ่งอันนี้สมมติว่าผู้อ่านได้ติดตั้ง board esp32 และ esp8266 ไว้เรียบร้อยแล้วนะครับ ขั้นแรกก็ Download/Clone Library จาก Github มาไว้ที่เครื่องก่อน โดยเข้าไปที่ Painlessmesh ที่ Github



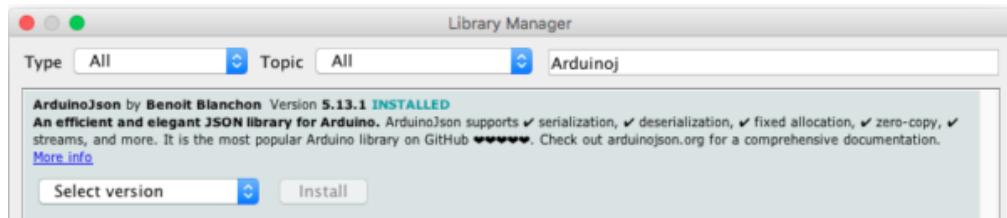
จากนั้นก็ติดตั้งลงใน Arduino IDE โดยเข้าไปที่ Sketch -> Include Library -> Add .zip library แล้วเลือก zip file ที่เรา Download มา



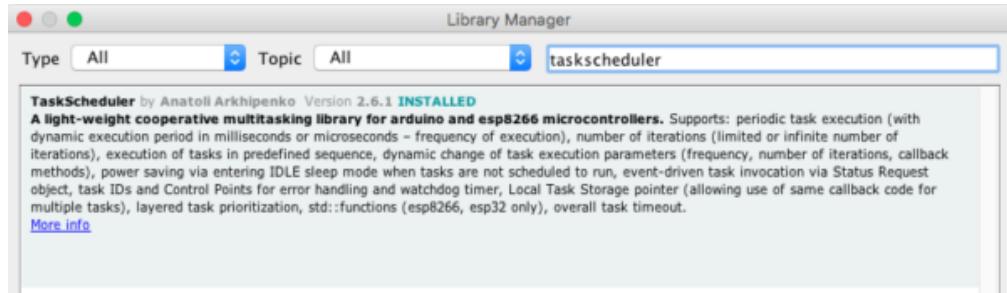
ติดตั้ง Library Dependencies ซึ่งจำเป็นต่อการใช้ Painlessmesh

ก่อนการใช้งานเรายังต้องใช้ Library อีกสองตัวนั้นก็คือ ArduinoJson และ TaskScheduler ซึ่งการ Install Library ก็สามารถทำได้ไม่ยากเลยสำหรับสองตัวนี้ เข้าไปที่ Sketch -> Include Library -> Manage Libraries จากนั้นในช่อง Filter Your Search ก็สามารถใส่ชื่อของ Library ทั้งสองตัวนี้ได้เลย

ArduinoJson



TaskScheduler



สิ่งที่ต้องรู้และข้อจำกัด

ทางผู้พัฒนา Painlessmesh เค้าเคลมเอาไว้ว่า library ชุดนี้นั้นเป็น True Adhoc Networking นั่นก็คือไม่ต้องมีการออกแบบว่า node ไหนจะเชื่อมต่อ node ไหน โครงสร้างของ mesh จะเป็นยังไง ไม่ต้องมี router ศูนย์กลาง ซึ่งแค่ 2 node ก็สามารถทำงานได้แล้ว ทำให้คนพัฒนาไม่ต้องมาจุ่งวุ่นวายในเรื่องของการจัดการกับ Self-Organize / Self-Configure ไปสนใจในส่วนของ Business Logic / Operation Logic ว่าจะใช้งาน Mesh นื้อย่างไรดีกว่า

ซึ่งคำถามของคนใช้ที่นักจะถามบ่อยๆ เรียกได้ว่าเป็น FAQ ของ Painlessmesh เลยก็ว่าได้นั่นก็คือจำนวน Maximum Node ที่รองรับ โดยที่ทางทีมผู้พัฒนา ก็บอกไว้ว่า

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

นั่นก็คือ ขึ้นอยู่กับ memory ของแต่ละ node นั่นเอง เพราะมันจะต้องใช้ในการเก็บสถานะของแต่ละ node เพื่อที่จะได้ทำ routing config ต่างๆ ฉะนั้นถ้าแต่ละ Node อ่านค่าจาก Sensor ตามรอบเวลาที่กำหนดแล้วส่งผ่าน Mesh Network ไปเก็บไว้ที่ Server จำนวน memory ที่ใช้ก็ไม่น่ามาก ซึ่งเดียวเราจะมาดูกันว่า memory นั้นจะถูกใช้ไปเท่าไหร่เมื่อเพิ่ม node เข้าไปใน Mesh Network

- non-ip networking: ในระบบ Network เราจะคุ้นชินกับระบบ IP หรืออย่างน้อยก็ mac address แต่ว่า Painlessmesh นั้นไม่ใช้ tcp/ip ในการสื่อสารและไม่ได้อ้างอิงโดยใช้ 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 อยู่แล้วก็จะง่ายเลยทีเดียว
- Delay: เริ่มเข้าสู่เรื่องข้อจำกัดละ ข้อจำกัดแรกเลยก็คือเรื่องของ delay ซึ่งผู้พัฒนา Painlessmesh เค้าแนะนำให้หลีกเลี่ยง เพราะภายในตัว library เองจะมีรองรับในการคุยกับ node ต่างๆ เพื่ออัปเดตสถานะของ mesh ตลอดเวลา node ไหนอยู่ node ไหนหลุดไป ถ้าเราเมื่อการใช้งาน delay ภายใน loop อาจทำให้ mesh ของเราไม่เสถียร ซึ่งทางแก้คือแนะนำให้ใช้ task scheduler แทน
- อัตราการส่งข้อมูล: ทางผู้พัฒนาเค้าแนะนำมาให้คิดคำนวณอัตราการส่งข้อมูลแบบ conservative ซะน้อย ต้องดูให้เหมาะสมกับ application ด้วย เช่นถ้าจะ monitor temp/humid ในดินแต่ส่งข้อมูลทุก 5 วินาทีก็อาจจะได้ไม่ค่อยสมเหตุสมผลเท่าไหร่ เพราะจะทำให้ตัว esp8266/esp32 ที่มีข้อจำกัดเรื่องของการประมวลผลและหน่วยความจำอาจต้องทำงานหนักขึ้น
- Message Dropped: ถึงแม้ระบบ Mesh Network มันจะเป็นอะไรที่เจ้มากจนดูเหมือนเป็น fault tolerance / high availability network แต่ก็มีโอกาสทำงานผิดพลาดได้ ฉะนั้น message ที่ส่งผ่านก็อาจจะหายไปได้บ้างในบางโอกาส ฉะนั้นก็ขึ้นอยู่กับ logic ของ application ที่เราพัฒนาว่าจะ handle ยังไง

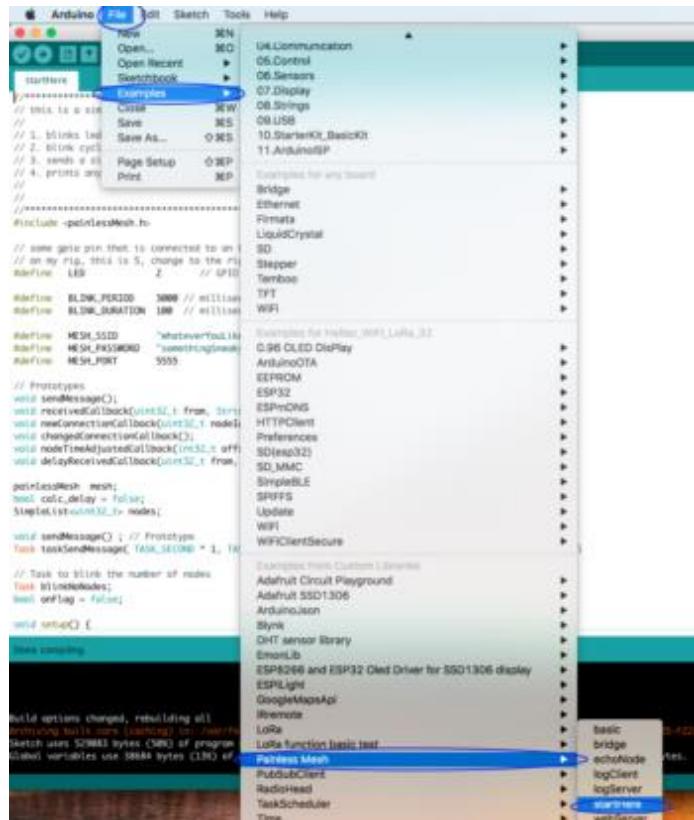
ໄอ้ำข้อกำหนดข้างบนหนะ ผูกกีไม่ได้เก่งเขียนขึ้นมาเองหรอกรับ ทีมพัฒนา Painlessmesh เนี่ยแหล่ะ แนะนำมา ผุกไว้ก็ยืดไว้เป็น best practice บ้างก็ได้รับ เวลาจะหาข้อผิดพลาดจะได้ไม่ต้องวนหาตั้งแต่ศูนย์ จะได้มาเช็คได้ว่าอี๊ที่มันส่งข้อมูลไม่ผ่านเป็นเพราะอะไร memory overflow มั้ย หรือเป็นปัญหาจากภัยภาพที่ไม่ join เข้า mesh หรือว่าเรามีเขียน delay ไว้

Hello Mesh Network

หลังจากเตรียม environment ในการทำงานเรียบร้อยแล้วเราก็จะมาเริ่มทดลองสร้าง Mesh Network แรกของเรากันนะอोเจ้า เขียนร่ายมาซะยาว เริ่มซัก 5 node ก่อนละกัน โดยที่ Node ของเราจะประกอบด้วย



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



มาดูการตั้งค่าคร่าวๆของ Mesh Network เรา กัน

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

จากไฟล์ startHere ที่มากับตัวอย่าง เราสามารถที่จะแยกวงของ Mesh รวมถึงเพิ่ม Security ในส่วนของ Mesh Network เราได้จาก 3 บรรทัดบนนี้เลยครับ ซึ่ง help ของ painlessmesh รวมถึงคำอธิบายที่อยู่ใน code นั้นค่อนข้างดีเลยที่เดียว ซึ่งการทำงาน Mesh Network โดยใช้ Painlessmesh นั้นจะแบ่งการทำงานเป็นลักษณะของ task โดยการกำหนดจาก task scheduler อย่างเช่น task ที่ใช้ในการส่ง message

```
mesh.scheduler.addTask( taskSendMessage );
taskSendMessage.enable();
```

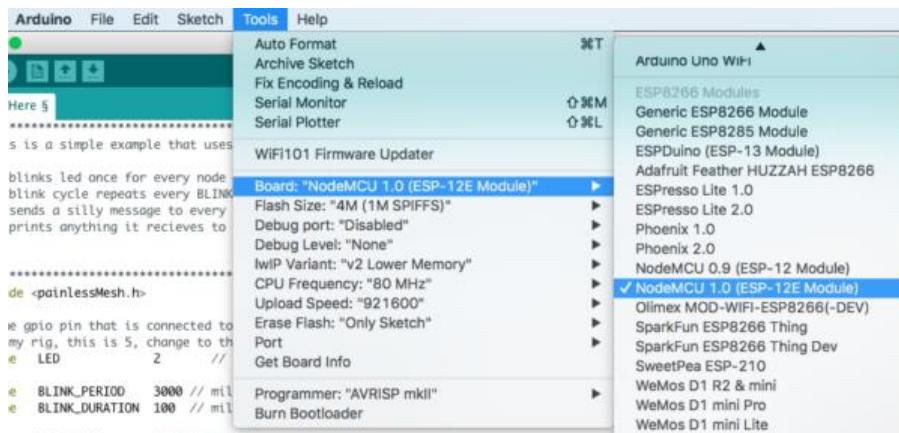
และในส่วนของ loop ก็จะมีแค่ การ update mesh เป็นหลัก พวก logic ต่างๆจะไปอยู่ที่ task scheduler หมด

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

ในส่วนของCallBack นั้นหลักที่ใช้งานจริงๆเลยก็คือ receivedCallback เพื่อที่จะดูว่า message ที่เราได้รับมา ไม่ว่าจะมาจาก Broadcast หรือ来自แบบระบบ NodeID นั้นเราจะนำไปทำอะไรต่อซึ่งในตัวอย่างนี้ก็คือ Print ออกมาผ่านทาง Serial Port

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

ซึ่งที่ผมแก้หลักๆเลยก็คือตัว SSID และ Password แค่นั้นจากนั้นก็ทำการ Flash เลยครับ เลือก Board ให้ตรงรุ่นแค่นั้น



ซึ่งจุดประสงค์หลักของตัวอย่างนี้ก็ตามด้านล่างนี้เลย กระพริบ led ตามช่วงเวลาที่กำหนด แล้วกีส่งข้อความแบบ broadcast ไปยังทุกๆ node ต้องลองทำตามครับตัวอย่างนี้ตัวอย่างเดียวได้อะไร酵อะ

```
// 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 ข้อมูลมันก็จะเยอะๆหน่อย เพราะมีการ enable debug ไว้ในตอน setup mesh.setDebugMsgTypes(ERROR | DEBUG | CONNECTION); ซึ่งระดับของการ debug ก็มีหลายแบบให้ลองเลือกใช้งานตามที่เหมาะสมดูครับ

```

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

```

ซึ่งเมื่อ flash เสร็จและเริ่มทำงาน nodemcu ตัวแรกในวง Mesh Network (มีตัวเดียวจะเรียก Mesh ที่มา yayy) ก็จะเริ่มหา AP ที่ซื้อเดียวกัน บ้านใกล้เรือนเคียงของ node อื่นๆ ละ เพื่อที่จะดึงมาเข้าสู่ Mesh นี้โดย ข้อมูลจากการ Debug ก็จะได้ nodeid รวมถึง memory ที่เหลืออยู่ยัง

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

```

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

```

คราวนี้เรามี Node ที่อยู่ใน Mesh นี้ 3 ตัวลงนั้นคือ 5851, 4414 และ 9003 ซึ่งแต่ละตัวก็จะ Broadcast Message อกกมาตามเวลาที่สุ่มได้ในแต่ละตัว แต่ในขั้นตอนการกระพริบไฟ Led นั้นจะมีการตั้ง time synchronize เมื่อมี node ใหม่เพิ่มเข้ามาใน mesh ซึ่งจะมีCallBack ไว้รองรับและทำงานร่วมกับ MeshUpdate ที่วน loop หาว่ามี Node ไหน drop connection ไปหรือมี node ไหน join เข้ามาและมีค่าความแรงของสัญญาณเท่าไหร่ เพื่อใช้ในการจัดเส้นทางในการส่งข้อมูล

```

/dev/cu.SLAB_USBtoUART
[REDACTED]
0x8 espWiFiEventCb(): SYSTEM_EVENT_AP_STA_CONNECTED
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(): 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: 33366
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

```

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

```

/dev/cu.SLAB_USBtoUART
[REDACTED]
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: 12268
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 = 356
Adjusted time 2278841890. Offset = 2064 Time Sync
Adjusted time 2278847262. Offset = -621
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

```

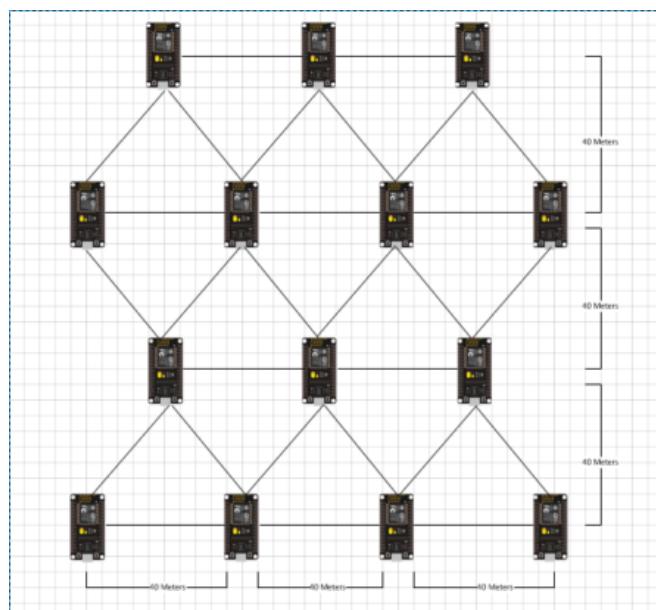
จากรูปด้านบนเมื่อทำงานไปได้ซักพักถ้าดู Memory ที่เหลืออยู่ต่ำสุด จะเป็นของ Nodemcu03 ซึ่งเหลืออยู่ 17408 Byte ส่วนที่เหลือมากสุดก็เดาไม่ยากเลย ย่อมเป็น ESP32 MCU ตัวใหม่แน่นอนอยู่แล้ว ดังที่ทางทีมผู้พัฒนาเค้าแจ้งเอาไว้ว่า mesh network นั้นจะรองรับได้กี่ node นั้นอยู่ที่ memory เป็นหลักเลย ไว้มีเวลาตอนที่ 2 จะเพิ่ม node เข้ามาใน mesh network พร้อมกับเพิ่มการส่งข้อมูลที่อ่านได้จากเซนเซอร์ต่างๆ แล้วมาดูกันว่า ขนาดที่เพิ่มขึ้นพร้อมกับ operation logic ที่เพิ่มขึ้น จะทำให้โครงข่าย mesh นี้เปลี่ยนไปยังไงกันบ้าง รับรองว่าตอนต่อไปจะได้นำ mesh network มาใช้งานในการส่งข้อมูลกันจริงๆ ละ ส่วนตอนต่อๆ ไปก็จะมีอะไรบ้างก็ลิสต์ขึ้นมาให้ดูกันเรียกแขกกันชะหน่อย

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

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

จาก ตอนที่ 1: Introduction & Painlessmesh ผู้อ่านน่าจะได้เข้าใจหลักการทำงานของ Mesh Network และการใช้งาน Library Painlessmesh ร่วมกับ ESP8266/ESP32 กันไปบางแล้วซึ่งเป็นการ Broadcast Message ไปยังทุก Node ที่อยู่ใน Mesh Network ซึ่งทั้งนี้ทั้งนั้นอย่างให้เข้าใจหลักการการใช้งาน กันก่อน เพราะถ้าเราเข้าใจหลักการแล้ว การประยุกต์ใช้งานจะทำได้ง่ายและเหมาะสมขึ้น จึงเราก็มาทวนกัน หน่อยละกันนะ

High Availability / Fault Tolerance

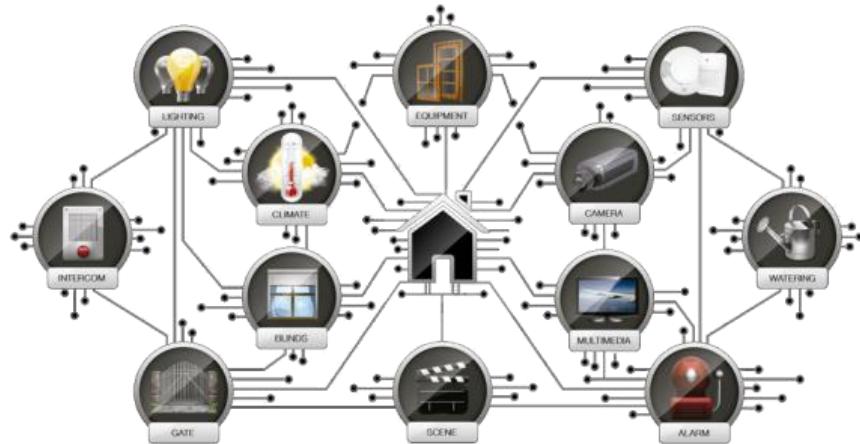


ขยายความจากตอนเดิม Mesh Network นั้นข้อดีก็คือเรื่องของความเสถียร สะดวก แทบไม่ต้อง setup อะไรเลย ถ้าดูตัวอย่างจากรูปด้านบน ถ้ามี Node ใด Node หนึ่งร่วงไปหรืออาจจะมากกว่าหนึ่งตัวก็ได้ เช่นทางในการส่งข้อมูลก็ยังคงไม่ถูกตัดขาด เพราะด้วยคุณสมบัติของ Self-Organize / Self-Configure ก็จะทำให้สามารถ ส่งข้อมูลไปยังปลายทาง หรือ Node ที่ยัง Active อยู่ได้ โดยไม่ต้องมีอุปกรณ์เป็นศูนย์กลางเหมือนระบบ WiFi ที่มี การทำงานแบบ Star ซึ่งถ้า WiFi Router หรือ AP ล่ม ทุก Node ก็จบ

Coverage Area

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

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



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

Power Consumption

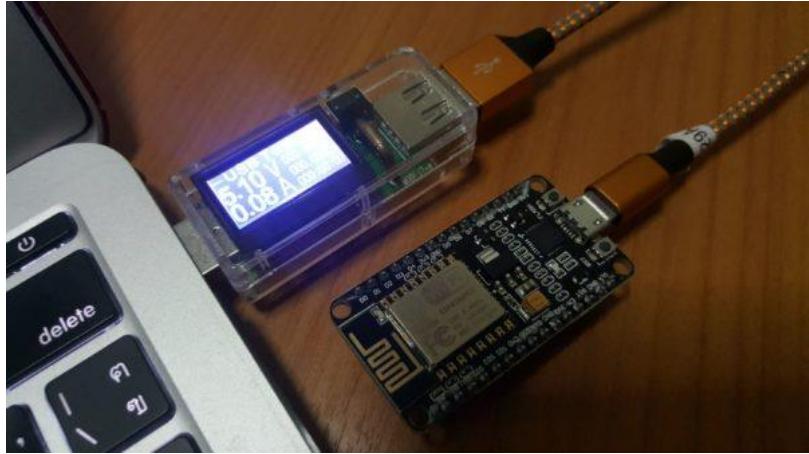
ถ้าจุดที่เราติดตั้งมีไฟฟ้าใช้งานตลอดเวลา หรือมีความสะดวกสบายในการเดินสายไฟ ประเด็นเรื่อง Power Consumption อาจจะไม่ต้องคิดมากนัก แต่ในระบบ home automation มักจะมีอุปกรณ์ประเภทที่ติดตั้งไว้โดยดูๆ หรือตอนที่บ้านสร้างเสร็จแล้ว อาจไม่สะดวกในเรื่องของการเดินสายไฟ ซึ่งอุปกรณ์เหล่านี้ก็จะเป็นพวก

- PIR ไว้ตรวจจับความเคลื่อนไหวต่างๆ
- Leak Sensor ไว้ดูว่ามีน้ำรั่วใต้ชิงค์ล้างจาน หรือใต้ห้องครัวมั้ย
- IR Remote ซึ่งทำให้เราสามารถที่จะควบคุมอุปกรณ์ต่างๆ เช่น TV แอร์ พัดลม

ที่ยกตัวอย่างไป ถ้าบ้านที่สร้างเสร็จไปแล้ว และต้องมาเดินสายไฟ อาจไม่สวยงามนัก Zigbee กับ Z-wave จึงมาตอบโจทย์ตรงนี้ ในการเชื่อมต่อ ก็เป็นลักษณะ Mesh Network ไม่ต้องเดินสาย แม่กินไฟต่ำ ใส่ถ่าน A23 ไปสองก้อนอยู่ได้เป็นปี ทำให้สะดวกมากๆ แต่อุปกรณ์เหล่านี้ก็มีราคาแพง ถ้าเทียบกับ ESP8266/ESP32

ซึ่งถ้าเราจะใช้ ESP8266/ESP32 แล้วหลังสิ่งหนึ่งที่ต้องคำนึงถึงเลยก็คือเรื่องของการจ่ายไฟให้ Node เหล่านี้ทำงานอยู่ตลอดเวลา ไม่สามารถที่จะ Sleep เพื่อที่จะประหยัดไฟได้ เราลองดูกันว่าในสถานการณ์ต่างๆ

เจ้า nodemcu ของเรา กินกระแสมากน้อยขนาดไหน ซึ่งลองวัดกันแบบหยาบๆ ผ่าน usb tester กัน ซึ่งถ้าใครมี Oscilloscope หรือ Multi-meter อยู่ใกล้ตัวกันน่าจะวัดได้ละเอียดขึ้น



Scenario1: rom เปล่าๆ วนลูปไปเรื่อย ค่าที่ได้ 80mA

Scenario2: ต่อ WiFi ทิ้งไว้ตามปกติ ค่าที่ได้อยู่ที่ 80mA

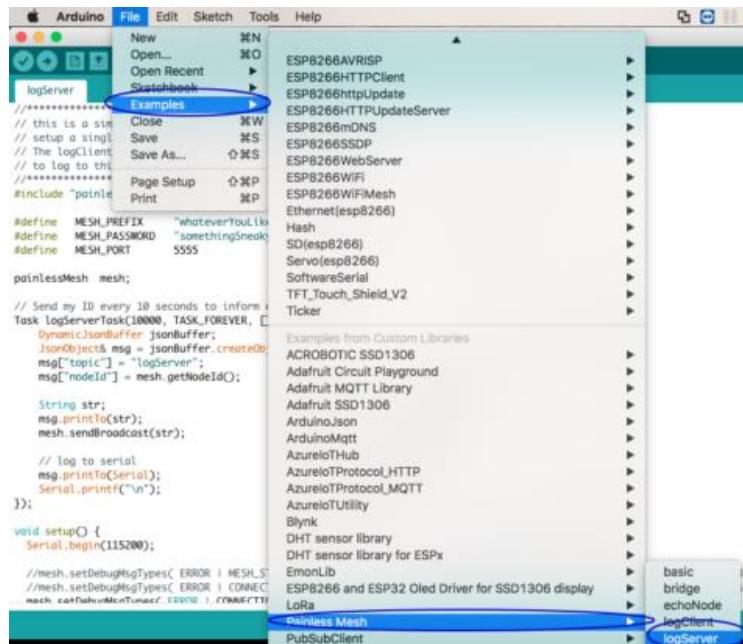
Scenario3: Join Mesh Network (ใช้ Painlessmesh) 2-6 Nodes Broadcast Message ค่าที่สังเกตุได้อยู่ที่ประมาณ 80-90mA มีค่า Peak อยู่ที่ประมาณ 150mA ซึ่งน่าจะมาจากการที่ Join/Update Mesh Network ซึ่งคาดว่าใช้งานจริงเพิ่ม Node เข้าไป น่าจะทำให้อัตราการกินกระแสสูงขึ้น ถ้าอยากรู้กระแสต่ำลงอาจจะต้องพิจารณาใช้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

ฉะนั้นถ้าจะเอา ESP8266/ESP32 มาทำ Mesh Network เพื่อรับส่งข้อมูล หรือ Control อุปกรณ์ต่างๆ แล้ว น่าจะต้องคำนึงในเรื่องของภาคจ่ายไฟด้วย ในโรงงานหรืออาคารที่ไม่มีปัญหาเรื่องการจ่ายไฟ น่าจะเหมาะสม ส่วนงานประเภท Outdoor / Open Field อาจต้องพิจารณาในเรื่องของการนำพา Solar Cell มาใช้ ก็เป็นอีกทางเลือกนึงที่น่าสนใจ ซึ่งจากข้อมูลด้านบนก็จะทำให้เราสามารถเตรียมในเรื่องของขนาด Solar Cell และ Battery สำรองไฟไว้ให้เหมาะสมได้ (อย่าลืมวัดจากการใช้งานใน application จริงด้วย เพราะเซนเซอร์บางอย่าง กินกระแสโดยเมื่อนกัน)

ในตอนที่ผ่านมาใครที่ผ่านมาอ่านแล้วได้ทดลองทำงานตัวอย่างแรกไป อาจเริ่มตัวอย่างที่สองในส่วนของการทำงานลักษณะที่เป็น Client/Server ไปแล้วก็ได้ เพราะตัวอย่างที่ให้มานั้นค่อนข้างดีเลยทีเดียว แต่ถ้าใครยังไม่ได้เริ่ม ก็มาเริ่มพร้อมๆ กันเลยครับ เกริ่นช่วยๆ อีกแล้ว

เริ่มใช้งาน Painlessmesh แบบ Client/Server

ตัวอย่างที่ประกอบสำหรับการใช้งานแบบ Client/Server จะใช้อ่ายส่องตัวด้วยกันคือ 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 ของเราซึ่งก็คือ 3 บรรทัดด้านบนนี้ครับ หรือใครจะคงเดิมไว้เหมือนตัวอย่างที่ให้มาก็ได้แล้วก็ Flash ลง ESP8266/ESP32 ของเราได้เลย

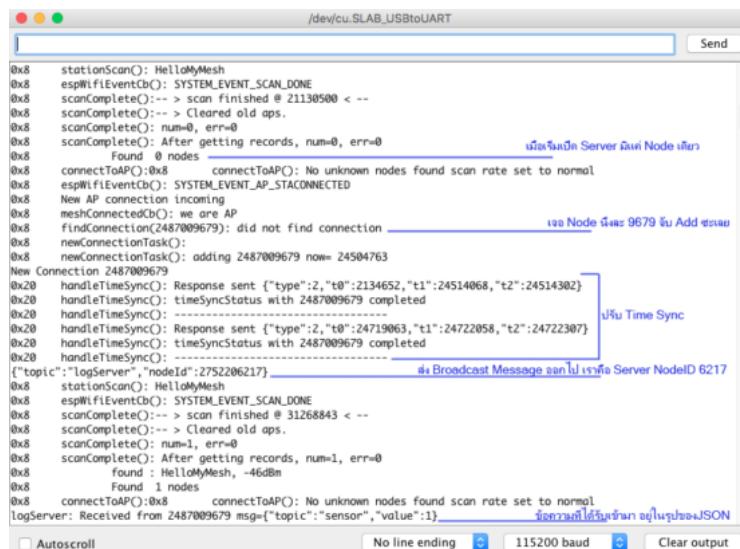
แต่ในส่วนของคนที่ใช้ ESP32 ต้องแก้ไขนิดนึงครับ เพราะการ Declare ในส่วนของ Authentication Mode นั้นไม่เหมือนกัน แต่ ESP8266 จะแก้ให้เป็นแบบ ESP32 ก็สามารถใช้งานได้เหมือนกันครับ

```
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 มีจุดประสงค์ในการส่งออกไปเพื่อบอกให้ node ที่อยู่ใน Mesh Network นี้รู้ว่า ฉัน NodeID นี้นั่นเป็น Server Node นะ ซึ่งก็จะทำงานด้วย task scheduler ทุกๆ 10 วิ ฉะนั้นมี node ใหม่เข้ามา join ใน Mesh Network ก็จะรู้แล้วว่าจะต้องส่งข้อมูลให้ครับซึ่งรูปแบบ Message ที่ส่งออกไปก็อยู่ในฟอร์แมทของ JSON ครับ

โดยในส่วนของ receivedCallback ก็ยังเหมือนเดิมคือ ได้รับ message มาก็ Print ออกทาง Serial Port ไม่มีการเปลี่ยนแปลง

```
Task logServerTask(10000, TASK_FOREVER, []){
    DynamicJsonBuffer jsonBuffer;
    JsonObject& msg = jsonBuffer.createObject();
    msg["topic"] = "logServer"; //กำหนด Topic เพื่อที่ Node ที่ join เข้ามาหาข้อมูลของ Server
    // เมื่อ
    msg["nodeId"] = mesh.getNodeID(); //ใช้คำสั่ง mesh.getNodeID(); เพื่ออ่านค่า NodeID
    String str;
    msg.printTo(str);
    mesh.sendBroadcast(str); // ส่งข้อความ Broadcast ไปยังทุก Node ว่าฉันคือ Server
    Node นะ
    // log to serial
    msg.printTo(Serial);
    Serial.printf("\n");
});
```



หลังจาก Flash เสร็จการทำงานก็เป็นอย่างที่เห็นข้างบนครับ จะเริ่มทำการเช็คข้างเคียงว่ามี Node ไหนที่สามารถ Join ได้บ้างจากนั้นก็ปรับ Offset Time Synchronize ให้เท่ากับเพื่อนๆในวง社群 แล้วก็เข้าสู่ Task ที่ใช้ในการ Broadcast Message ออกไปว่าเราคือ Server เพื่อให้ Node ทุกตัวที่อยู่ใน Mesh Network เดียวกันรู้ว่าจะส่งข้อมูลให้ Server ให้ส่งไปที่ NodeID หมายเลขอะไร ซึ่งในที่นี้ก็คือ 6217

มาตรฐานส่วนของ logClient กันบ้าง ซึ่งในส่วนของ receivedCallback เนี่ยแหล่ะที่จะเปลี่ยนไป เพราะจะมีการเช็ค Broadcast Message จาก Node อื่นๆ ซึ่งก็คือ Server Node ว่า เห้วยย นี่แหล่ะ Server Node นะ มี ID นั่นจะ จากนี้ถ้าจะส่งข้อความไปที่ ServerNode ให้ส่งไปที่ NodeID นั่นจะ

```

1. void receivedCallback( uint32_t from, String &msg ) {
2.     Serial.printf("logClient: Received from %u msg=%s\n", from, msg.c_str());
3.
4.     // Saving logServer
5.     DynamicJsonBuffer jsonBuffer;
6.     JsonObject& root = jsonBuffer.parseObject(msg);
7.     if (root.containsKey("topic")) {
8.         if (String("logServer").equals(root["topic"].as<String>())) { // เช็ค Topic ว่า
9.             เป็น logServer หรือเปล่า
10.            // check for on: true or false
11.            logServerId = root["nodeId"]; // อ่านและบันทึกค่า nodeID ของ Server เพื่อใช้เป็น
12.            // ปลายทางในการส่ง
13.            Serial.printf("logServer detected!!!\n");
14.        }
15.    }

```

สำหรับส่วนของ Task ที่ใช้ในการส่ง Message ก็จะทำงานล้อกัน คือถ้ายังไม่เจอว่า ใน Mesh Network ที่เพิ่ง join เข้ามา มี server หรือเปล่า ฉันก็ broadcast ข้อมูลของฉันเลยละกัน เนื่องจาก Server Node มีระยะเวลา 10 วินาทีถึงจะ Broadcast Message Topic “logserver” ออกไป ฉะนั้นการทำเช่นนี้ Server Node ที่รู้ข้อมูลจาก Client Node ก็จะได้ข้อมูลแน่นๆ (รวมถึงตัวอื่นๆด้วย)

```

Task myLoggingTask(10000, TASK_FOREVER, []() {
    DynamicJsonBuffer jsonBuffer;
    JsonObject& msg = jsonBuffer.createObject();
    msg["topic"] = "sensor";
    msg["value"] = random(0, 180); // สำหรับไฟล์ตัวอย่างนี้ยังไม่ได้ต่อเข้ากับเซ็นเซอร์อะไร
    ก็สุ่มค่าส่งไปเลยกัน
    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 espWifileventCb(): SYSTEM_EVENT_SCAN_DONE
0x8 scanComplete():--> scan finished @ 21204468 <--
0x8 scanComplete():--> Cleared old aps.
0x8 scanComplete(): 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 espWifileventCb(): SYSTEM_EVENT_STA_AUTHMODE_CHANGE
0x8 espWifileventCb(): SYSTEM_EVENT_STA_CONNECTED
0x8 espWifileventCb(): 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 ถ้าต้องการรับข้อมูลจาก Log Server ให้ตั้งค่า Serve NodeID
logServer detected!!!
Handled from 2752206217 msg={"topic": "logServer", "nodeId": 2752206217} ผู้ให้บริการ Log Server ที่ต้องการรับข้อมูล Topic Server ให้ตั้งค่า Serve NodeID
{"topic": "sensor2", "value": 2}
Received from 2752206217 msg={"topic": "logServer", "nodeId": 2752206217} Broadcast Message ถ้าต้องการรับข้อมูลจาก Log Server ให้ตั้งค่า Serve NodeID
logClient: Received from 2752206217 msg={"topic": "logServer", "nodeId": 2752206217}
logServer detected!!!
Handled from 2752206217 msg={"topic": "logServer", "nodeId": 2752206217} ผู้ให้บริการ Log Server ที่ต้องการรับข้อมูล Topic Server ให้ตั้งค่า Serve NodeID
{"topic": "sensor2", "value": 149}

```

คราวนี้ແທນที่เราจะจำลองข้อมูลเมื่อวันในตัวอย่าง ก็ได้เวลาเปิดกระปองแระ เอา DHT22 มาพ่วงเข้าไป
คราวนี้ Temp/Humid ที่ได้ก็จะเป็นค่าจริงแล้วโดยการปรับ Code ในส่วนของ Task ที่ใช้สำหรับการส่งข้อความ
ซึ่งจะมีการอ่านค่าจาก DHT22 แล้วจัดรูปแบบให้อยู่ในฟอร์แมทของ JSON โดยที่ Mesh Network นี้จะประกอบ
ไปด้วย 12 Node ด้วยกันดังนี้



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

Code ที่ทำการแก้ไขสำหรับ Node ตัวที่สูนส่งค่า (ESP-01, Wemos-01 ลีง 04, mcu-01 ลีง 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 เข้ามาพอยท้ายๆตัว
    เริ่มงกับ เลข NodeID
    msg["NodeID"] = mesh.getNodeId();    //เพิ่ม NodeID เอาไว้อ้างอิง
    msg["random value"] = random(0, 180); //เปลี่ยนเป็น random value ให้ชัดเจน
    //ว่าเป็นค่าที่สุ่มขึ้นมา
    String str;
    msg.printTo(str);
    if (logServerId == 0) // If we don't know the logServer yet
        mesh.sendBroadcast(str);
    else
        mesh.sendSingle(logServerId, str);
    // log to serial
    msg.printTo(Serial);
    Serial.printf("\n");
});
```

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

```

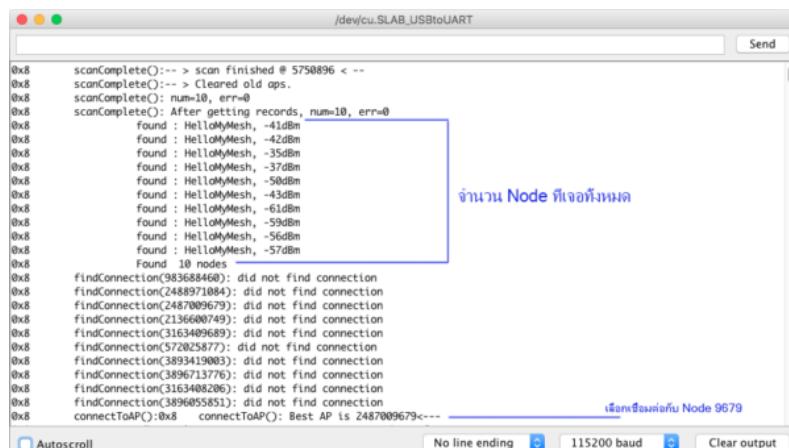
1. #include "painlessMesh.h"
2. #include "DHT.h"
3.
4. #define DHTPIN D4
5. #define DHTTYPE DHT22
6.
7. #define MESH_PREFIX "HelloMyMesh"
8. #define MESH_PASSWORD "hellomymeshnetwork"
9. #define MESH_PORT 5555
10.
11. DHT dht(DHTPIN, DHTTYPE);
12.
13. void receivedCallback( uint32_t from, String &msg );
14.
15. painlessMesh mesh;
16.
17. size_t logServerId = 0;
18.
19. // Send message to the logServer every 5 seconds
20. Task myLoggingTask(5000, TASK_FOREVER, []() {
21.
22.     float h = dht.readHumidity();
23.
24.     float t = dht.readTemperature();
25.
26.     DynamicJsonBuffer jsonBuffer;
27.     JsonObject& msg = jsonBuffer.createObject();
28.     msg["nodename"] = "Wemos-t1";
29.     msg["NodeID"] = mesh.getNodeId();
30.     msg["Temp"] = String(t)+"C";
31.     msg["Humidity"] = String(h)+"%";
32.
33.     String str;
34.     msg.printTo(str);
35.     if (logServerId == 0) // If we don't know the logServer yet
36.         mesh.sendBroadcast(str);
37.     else
38.         mesh.sendSingle(logServerId, str);
39.
40.     // log to serial
41.     msg.printTo(Serial);
42.     Serial.printf("\n");
43. });
44.
```

```

45. void setup() {
46.
47.   Serial.begin(115200);
48.   Serial.println("Begin DHT22 Mesh Network test!");
49.
50.   dht.begin();
51.
52.   mesh.setDebugMsgTypes( ERROR | STARTUP | CONNECTION ); // set before init() so
that you can see startup messages
53.
54.   mesh.init( MESH_PREFIX, MESH_PASSWORD, MESH_PORT, STA_AP, AUTH_WPA2_PSK, 6 );
55.   mesh.onReceive(&receivedCallback);
56.
57.   // Add the task to the mesh scheduler
58.   mesh.scheduler.addTask(myLoggingTask);
59.   myLoggingTask.enable();
60.
61. }
62.
63. void loop() {
64.   // put your main code here, to run repeatedly:
65.
66.   mesh.update();
67.
68.
69. }
70.
71. void receivedCallback( uint32_t from, String &msg ) {
72.   Serial.printf("logClient: Received from %u msg=%s\n", from, msg.c_str());
73.
74.   // Saving logServer
75.   DynamicJsonBuffer jsonBuffer;
76.   JsonObject root = jsonBuffer.parseObject(msg);
77.   if (root.containsKey("topic")) {
78.     if (String("logServer").equals(root["topic"].as<String>())) {
79.       // check for on: true or false
80.       logServerId = root["nodeId"];
81.       Serial.printf("logServer detected!!!\n");
82.     }
83.     Serial.printf("Handled from %u msg=%s\n", from, msg.c_str());
84.   }
85. }

```

หลังจาก Flash ครบตัวสุดท้ายก็เปิด Serial Monitor ขึ้นมาจะเห็นได้จากรูปข้างล่างเลยครับว่า เจอบรอดำเนินการด้วยตัวเอง Node ทั้งหมด 10 Node พร้อมกับความแรงของสัญญาณจากแต่ละ Node ซึ่งจะใช้ในการจัดการการเชื่อมต่อของ Mesh Network ของเรา



อย่างที่ได้บอกไปข้างต้นครับ เมื่อเริ่มการทำงาน Server จะ Broadcast Message เพื่อบอกเพื่อนๆใน Mesh Network นี้ว่า NodeID ของ Server นั้นคือหมายเลขอะไร เพื่อที่ว่าเมื่อไบ昂ครั้ง ได้มีการเพิ่มนод子เข้าไปใน Mesh Network นี้ก็สามารถที่จะรู้ได้ทันทีว่าจะต้องส่งข้อมูลไปที่ Server NodeID ไหน แต่ระหว่างที่ยังไม่รู้ว่า Server NodeID คือหมายเลขอะไร Client Node ก็จะ Broadcast Message ออกไปเพื่อแจ้งไปก่อน อย่างน้อย

มันก็ต้องเข้า Server Node และ คราวนี้เรามาจำลองเคสที่ว่ามีกันดู ระหว่างที่ยังไม่เจอ Server Node และเมื่อ Server Node Online และทุกๆ Node รับทราบกันหมดแล้วและส่ง Message อกไปแบบระบุ NodeID

Scenario ที่ 1: No Server Node

ตามรูปด้านล่างนี้เลยครับ ข้อมูลที่ได้รับผ่าน receivedCallback ได้รับข้อมูลจาก Node ที่ Online ใน Mesh Network นี้ทั้งหมด เพราะว่ายังไม่ได้เปิด Server

```

/dev/cu.SLAB_USBtoUART
Send
mcu-04 สถานะที่ 0 สถานที่ที่ 0
[{"nodeid": "mcu-04", "nodeid": "3896664414", "random value": 87}
logClient: Received from 3163409689 msg={"nodeid": "Wemos-02", "nodeid": "3163409689", "random value": 95}
logClient: Received from 3896055851 msg={"nodeid": "mcu-02", "nodeid": "3896055851", "random value": 55}
logClient: Received from 3896713776 msg={"nodeid": "mcu-01", "nodeid": "3896713776", "random value": 155}
logClient: Received from 3163408206 msg={"nodeid": "Wemos-04", "nodeid": "3163408206", "random value": 75}
logClient: Received from 3893419003 msg={"nodeid": "mcu-03", "nodeid": "3893419003", "random value": 2}
logClient: Received from 572025877 msg={"nodeid": "Wemos-01", "nodeid": "572025877", "random value": 148}
logClient: Received from 2487009679 msg={"nodeid": "Wemos-11", "nodeid": "2487009679", "Temp": "25.10C", "Humidity": "44.70%"}
logClient: Received from 2136600749 msg={"nodeid": "Wemos-03", "nodeid": "2136600749", "random value": 16}
logClient: Received from 2488971084 msg={"nodeid": "mcu-t1", "nodeid": "2488971084", "Temp": "25.90C", "Humidity": "42.10%"}
logClient: Received from 983688460 msg={"nodeid": "esp-01", "nodeid": "983688460", "random value": 96}
logClient: Received from 3163409689 msg={"nodeid": "mcu-02", "nodeid": "3163409689", "random value": 142}
logClient: Received from 3163409689 msg={"nodeid": "Wemos-02", "nodeid": "3163409689", "random value": 149}
logClient: Received from 3896055851 msg={"nodeid": "mcu-02", "nodeid": "3896055851", "random value": 37}
logClient: Received from 3896713776 msg={"nodeid": "mcu-01", "nodeid": "3896713776", "random value": 133}
logClient: Received from 3163408206 msg={"nodeid": "Wemos-04", "nodeid": "3163408206", "random value": 176}
logClient: Received from 3893419003 msg={"nodeid": "mcu-03", "nodeid": "3893419003", "random value": 132}
logClient: Received from 572025877 msg={"nodeid": "Wemos-01", "nodeid": "572025877", "random value": 91}
logClient: Received from 2136600749 msg={"nodeid": "Wemos-03", "nodeid": "2136600749", "random value": 13}
logClient: Received from 2487009679 msg={"nodeid": "Wemos-11", "nodeid": "2487009679", "Temp": "25.40C", "Humidity": "44.88%"}
logClient: Received from 2488971084 msg={"nodeid": "mcu-t1", "nodeid": "2488971084", "Temp": "25.90C", "Humidity": "42.18%"}
logClient: Received from 983688460 msg={"nodeid": "esp-01", "nodeid": "983688460", "random value": 106}
{"nodeid": "mcu-04", "nodeid": "3896664414", "random value": 26}
logClient: Received from 3163409689 msg={"nodeid": "Wemos-02", "nodeid": "3163409689", "random value": 27}
logClient: Received from 3896055851 msg={"nodeid": "mcu-02", "nodeid": "3896055851", "random value": 61}

No line ending 115200 baud Clear output
Autoscroll

```

Scenario ที่ 2: Server Node join Mesh Network

เมื่อเปิด Server Node เข้ามา ก็จะเป็นเหมือนรูปข้างล่างนี้ครับ ได้รับ Broadcast Message จาก Server Node และในส่วนของ receivedCallback ก็จะมี logic เพื่อใช้ในการอ่านค่า่ว่ามี Topic ที่เป็น logServer มั้ย ถ้ามีก็บันทึกเก็บไว้ เพื่อในการส่งรอบต่อไปจะไม่ Broadcast ไปยังทุก Node ละ แต่จะระบุปลายทางเป็น NodeID ของ Server แทน

```

/dev/cu.SLAB_USBtoUART
Send
mcu-04 สถานะที่ 0 สถานที่ที่ 0
[{"nodeid": "mcu-04", "nodeid": "3896664414", "random value": 77}
logClient: espNfEventCb(): SYSTEM_EVENT_AP_STACONNECTED
0x8 New AP connection incoming
0x8 meshConnectedCb(): we are AP
0x8 findConnection(2752206217): did not find connection
newConnectionTask():
newConnectionTask(): adding 2752206217 now: 3889582447
logClient: Received from 3893419003 msg={"nodeid": "mcu-03", "nodeid": "3893419003", "random value": 163}
logClient: Received from 572025877 msg={"nodeid": "Wemos-01", "nodeid": "572025877", "random value": 1}
logClient: Received from 2136600749 msg={"nodeid": "Wemos-03", "nodeid": "2136600749", "random value": 49}
logClient: Received from 983688460 msg={"nodeid": "esp-01", "nodeid": "983688460", "random value": 20}
{"nodeid": "mcu-04", "nodeid": "3896664414", "random value": 77}
logClient: Received from 3163409689 msg={"nodeid": "Wemos-02", "nodeid": "3163409689", "random value": 30}
logClient: Received from 3896055851 msg={"nodeid": "mcu-02", "nodeid": "3896055851", "random value": 170}
logClient: Received from 3896713776 msg={"nodeid": "mcu-01", "nodeid": "3896713776", "random value": 128}
logClient: Received from 2488971084 msg={"nodeid": "mcu-t1", "nodeid": "2488971084", "Temp": "24.60C", "Humidity": "36.60%"}
logClient: Received from 3163408206 msg={"nodeid": "Wemos-04", "nodeid": "3163408206", "random value": 37}
logClient: Received from 3893419003 msg={"nodeid": "mcu-03", "nodeid": "3893419003", "random value": 174}
logClient: Received from 572025877 msg={"nodeid": "Wemos-01", "nodeid": "572025877", "random value": 8}
logClient: Received from 2136600749 msg={"nodeid": "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"}
{"nodeid": "mcu-04", "nodeid": "3896664414", "random value": 81}

No line ending 115200 baud Clear output
Autoscroll

```

หลังจากที่ทุก Node ได้รับ Server NodeID จาก Broadcast Message ที่ส่งมาจาก Server Node แล้ว แทนที่ทุก Node จะส่งข้อมูลอกไปยังทุกๆ Node ก็จะส่งไปยัง Server Node แทนดังจะเห็นได้จากรูปข้างล่างที่

ได้รับ Broadcast Message เข้ามานะมีเฉพาะข้อความที่มาจากการ Server Node เท่านั้น เพราะตัวอื่นๆส่งไปที่ Server Node หมดแล้ว

```

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}

```

กรณีเรามาลองดูในฝั่งของ Server กันบ้างว่าเมื่อทุก Node ส่งข้อมูลมาที่ Server Node แล้วข้อมูลที่ได้จะเป็นลักษณะไหนกัน จากรูปจะเห็นได้ว่า ข้อมูลจะได้ครบเหมือนตอนที่เราดูของ Node ใด Node หนึ่งตอนที่ยังไม่มี Server Node เนื่องจากกรณีทุก Node ใน Mesh Network ส่งมาที่จุดเดียว ก็เลยจะได้ข้อมูลครบและตัว Server Node เองก็ยังคง Broadcast Message ออกไปเรื่อยๆเพื่อให้ทุกคนรู้ว่า ฉันอยู่ที่นี่นะ ถ้าจะส่งข้อมูลมาหา ให้ส่งมาที่ไหน

```

LogServer: Received from 2487009679 msg={"nodename":"Wemos-t1","NodeId":2487009679,"Temp":"23.60C","Humidity":"38.20%"}
LogServer: Received from 2136408296 msg={"nodename":"Wemos-04","NodeId":3163408296,"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 2136600749 msg={"nodename":"Wemos-03","NodeId":2136600749,"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":180}
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-01","NodeId":2487009679,"Temp":"23.60C","Humidity":"38.20%"}
LogServer: Received from 3896664414 msg={"nodename":"mcu-04","NodeId":3896664414,"random value":97}
LogServer: Received from 2136600749 msg={"nodename":"Wemos-03","NodeId":2136600749,"random value":67}
LogServer: Received from 2488971084 msg={"nodename":"mcu-t1","NodeId":2488971084,"random value":153}
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":184}
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":58}
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 Broadcast NodeID อยู่ที่นี่
LogServer: Received from 2487009679 msg={"nodename":"Wemos-t1","NodeId":2487009679,"Temp":"23.60C","Humidity":"38.20%"}
LogServer: Received from 2136600749 msg={"nodename":"Wemos-03","NodeId":2136600749,"random value":21}

```

ขยายความในส่วนของ Server Node กันอีกนิดว่าทำไม่ต้องคอย Broadcast Server NodeID อุํ

ตลอดเวลา

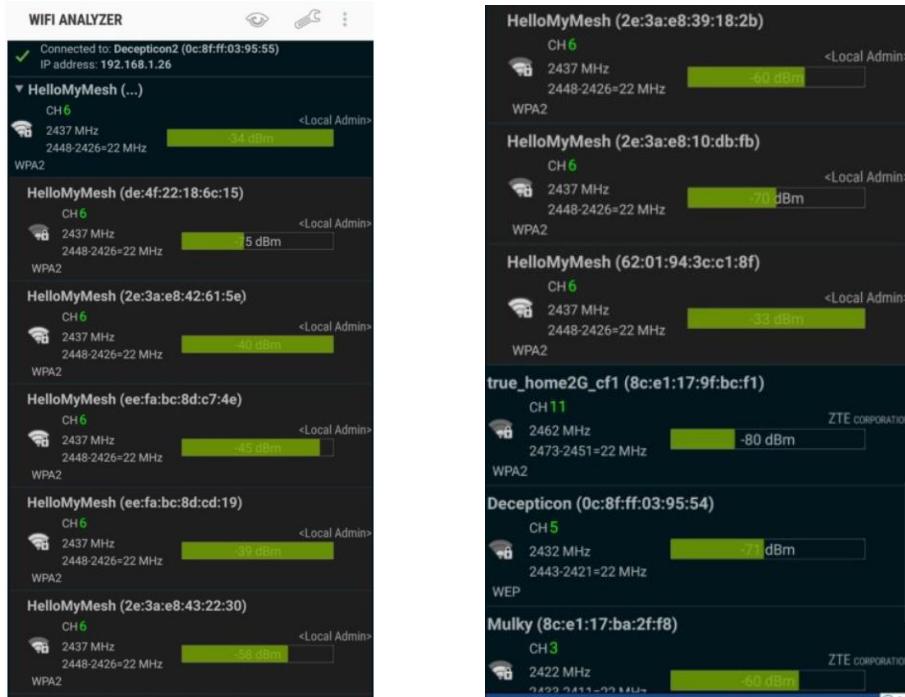
- ข้อดี Leyk็คือ ถ้าในอนาคตต้องมีการเปลี่ยนอุปกรณ์ในส่วนของ Server Node ที่เสียไป แต่เมื่อสลับ อุปกรณ์เปลี่ยน Server Node แล้วอนุว่า ChipID เปลี่ยนไปละ การ Broadcast Server NodeID และในส่วนของ Client ที่มี receivedCallback คอยเช็ค Server NodeID ก็จะทำให้แทบไม่กระทบ กับการทำงานเลย เพราะในทุกสุดทุก Node ก็จะรู้ว่าต้องส่งไปที่ Server Node ใหม่หมายเลขอะไร
- ซึ่งถ้าเข้าใจหลักการที่เขียนมาสองบทนี้ เราสามารถประยุกต์ใช้งานในการทำ Server Node ที่มีมากกว่า 1 ก็ได้ ยิ่งทำให้ระบบ Mesh Network ที่ใช้ในการสื่อสารออกไป Network ภายนอกนั้น เสถียรยิ่งขึ้น หรือจะแบ่งเบาภาระของ Server Node เป็นลักษณะของ load balance ก็ยังได้

ตัวอย่างการประยุกต์ใช้งาน

- Read and Forward: เคสนี้จะเข้ามายำหรับพ่วงการส่งข้อมูลเข้า Server อาจจะเป็นการวัด อุณหภูมิ ความชื้น หรือแม้กระทั่งเสียงบริเวณต่างๆของโรงงาน แล้วมาเก็บไว้ที่ Server เพื่อที่จะแสดงผล หรือนำข้อมูลไปวิเคราะห์เหตุต่างๆ เมื่อในตัวอย่าง logServer/logClient ที่เราเพิ่งทำ กันไป เป็นการใช้งานที่ค่อนข้าง One-way จะเป็นส่วนใหญ่ และก็เป็น Application ของ IoT จะเป็น ส่วนใหญ่อีก เช่นกันด้วย
- เนื่องจากการส่ง message ภายใน Mesh Network นี้สามารถที่จะกำหนดปลายทางผู้รับได้ด้วย NodeID หรือจะส่ง Broadcast แล้วค่อย Filter Message เอกก์ได้ว่าปลายทางที่ระบุใน Broadcast Message เป็นของเราระหว่างเปล่า จากนั้นอ่านค่าที่ได้มา เปรียบเทียบ logic ที่ต้องการ อาจจะเป็นการ สั่งให้เปิดปั๊มน้ำเพื่อรดน้ำต้นไม้ หรือสั่งให้เปิด Siren นอกอาคารกรณีมีการตรวจจับควันไฟได้ จะแจ้ง เป็น zone หรือแจ้งเป็นจุดๆ ก็แล้วแต่หลักการเขียนข้อความที่ส่งไป และหลักการเปรียบเทียบ logic ของข้อความที่ได้รับมา ซึ่ง Painlessmesh ใช้รูปแบบการส่งแบบ JSON ก็สะดวกยิ่งขึ้น
- Point to Point / Independent: เนื่องจาก Mesh Network นั้นทำให้ทุก Node นั้นเชื่อมหากันอยู่ แล้ว Node ไหน อยากคุยกับ Node ไหนก็ได้ เคลลักษณะนี้ก็อาจจะมี Node ที่ทำงานเป็นลักษณะ Control Panel สามารถที่จะเลือกได้ว่าจะส่งคำสั่งไปให้ node ตัวไหนก็ได้ โดยการอ้างอิงจาก NodeID หรืออาจจะแปลง NodeID ให้เป็นชื่อที่อ่านแล้วเข้าใจง่ายเหมือนที่ผมทำเป็นต้น

หมายเหตุ

เนื่องจาก ESP8266/ESP32 ยังใช้ WiFi ย่าน 2.4 GHz อยู่ ซึ่งเวลาใช้งานในโหมดของ Station and AP เพื่อให้ Node ต่างๆมา AGREEMENT กันเป็น Mesh Network แล้ว ถ้าเปิดโปรแกรมพวก WiFi Analyzer มาดูก็จะเห็น ชื่อ AP ใน Mesh ของเรามาเพียบเลย ซึ่งอาจจะไปกรุณากับ AP อื่นหรือ AP อื่นอาจจะมีกวนเรา ก็ได้ ยังไง ก็ลองพิจารณาการใช้งานให้เหมาะสมสมดุครับ



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

ESP8266 / ESP32 & Mesh Network ตอนที่ 3: Painlessmesh Bridge

ผ่านมาสองตอนเรียบร้อยกับเรื่องของ ESP8266/ESP32 ในการนำมาทำ Mesh Network ซึ่งยาวมากๆ ถ้าใครทำตามผ่านมาได้ถึงตอนนี้ไม่มีอะไรากลังครับ ตอนนี้จะเป็นเรื่องของการเชื่อมต่อหรือ Bridge เจ้า Local Mesh Network ของเราเพื่อออกไปสู่ Internet กันครับ ซึ่งในที่นี่เราจะเพิ่ม Bridge Node ของเราเข้าไป เพื่อต่อเข้ากับ Server Node แต่ก่อนอื่น เรา มาดูเรื่องการส่งข้อมูลผ่าน Serial Port ของ Nodemcu สองตัวกันครับ เพราะสามารถประยุกต์ใช้งานอย่างอ่อนน้อมั่นได้มาก

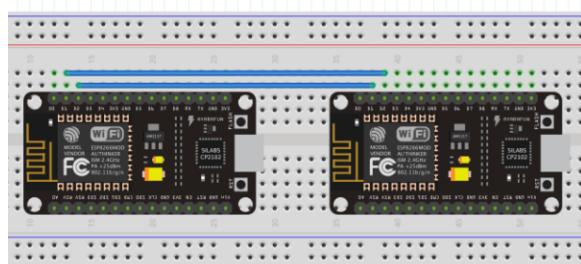
ตัวอย่างเพื่อประยุกต์ใช้ในการส่งผ่านข้อมูลของ Nodemcu สองตัวผ่าน Serial Port

Redundant (Active-Standby) : อย่างที่ทราบกันดีว่าพลังในการประมวลผลและหน่วยความจำของ ESP8266 นั้นมีจำกัด ซึ่งเราสามารถที่จะให้ Nodemcu ของเราทำงานสลับกันได้ กรณีตัวได้ตัวหนึ่งตายไป อีกตัวก็ขึ้นมาทำงานแทนแล้วก็ส่งสัญญาณไป reset Nodemcu ตัวที่ตายไป พอดีนปลูกขึ้นมาใหม่ก็มาอยู่ใน Standby Mode ค่อยฟังสัญญาณผ่านทาง Serial Port ซึ่งเราสามารถทำ Heart Beat Check ผ่านทาง Serial Port ได้ โดยที่ Nodemcu สามารถทำงานแทนกันไปแทนกันมาได้ตลอด เพราะอย่างที่ทราบกันดีอุปกรณ์ดีขนาดไหนก็มีโอกาสที่จะ hang ได้ ถ้า reboot แล้วกลับมาทำงานตามปกติก็ได้ แต่ถ้าไม่ปกติก็อาจทำให้งานที่ทำอยู่เสียหายได้ ถ้าเป็นสถานที่ที่ต้องเดินทางก็คงลำบากอีก บางคราวอาจจะคิดว่าอื้าawan ถ้าเงินก็ให้มันทำงานสองตัว

ตั้งแต่แรกก็ได้สิ ทำงานพร้อมๆกันเลย อันนี้อาจต้องมองแล้วแต่เครื่องรับ เพราะบางที่เราอาจจะไม่ได้ใช้งานเป็น Sensor Node อย่างเดียว

Communication Channel : เคสนี้จะเป็นเคสที่เราจะใช้กันในตอนที่ 3 นี้ครับ ซึ่งบางโปรเจคที่ทำกับ Arduino หรือ MCU บางตัวก่อนหน้านี้ที่ไม่มี WiFi รองรับแล้ววันนี้อยาจจะส่งค่าอุปกรณ์เพื่อ monitor ผ่านทาง Internet ก็สามารถทำได้โดยที่ MCU เดิมส่งค่าผ่านมาทาง Serial Port และใช้ ESP8266 รับค่า จากนั้นก็ส่งข้อมูลผ่าน WiFi

น่าจะพอเห็นภาพการใช้งานเพื่อสื่อสารระหว่าง Nodemcu 2 ตัวผ่าน Serial Port กันแล้ว เรามาดูในส่วนของ Code กันบ้างซึ่งเนื่องจากข้อจำกัดของ Nodemcu ที่มี Serial Port แค่คู่เดียว และเราใช้มันในการ Flash Code ของเราร่วมถึง Monitor ผ่าน Serial Monitor ใน Arduino IDE อีก แต่ว่าโชคดีที่ ESP8266 นั้นสามารถใช้งาน Software Serial เพื่อใช้ I/O ที่มีอยู่ในการทำงานเป็น Serial Port แทนได้ เพื่อให้เห็นภาพเรามาเริ่มที่ตัวอย่างนี้กันครับ โดยใช้ mcu2 ตัว



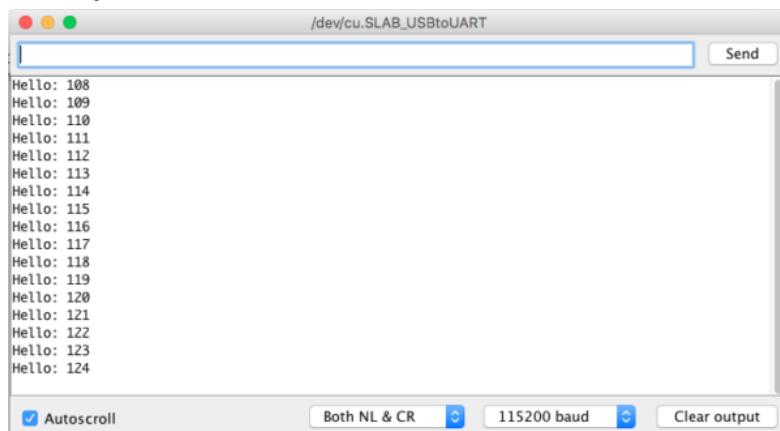
เริ่มจาก Copy Code ด้านล่างนี้แล้วก็ Flash ลง Nodemcu ทั้งสองตัวเลย ซึ่งใน Code ผมกำหนดให้ D1 เป็น RX ส่วน D2 เป็น TX ฉะนั้นการต่อสายจะเป็นลักษณะไขว้กันระหว่าง RX<->TX ตามรูปด้านบน

```

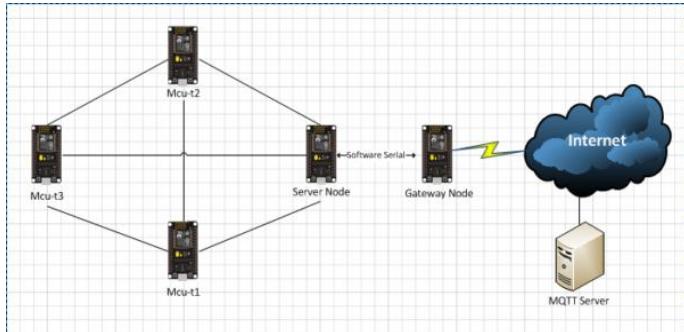
1. #include <SoftwareSerial.h>
2.
3. SoftwareSerial swSerial(D1, D2, false, 128); //Define hardware connections RX, TX
4.
5. int i = 0;
6. String DataString;
7.
8. void setup() {
9.
10. pinMode(D1, INPUT);
11. pinMode(D2, OUTPUT);
12.
13. Serial.begin(115200); //Initialize hardware serial with baudrate of 115200
14. swSerial.begin(115200); //Initialize software serial with baudrate of 115200
15.
16. Serial.println("\nSoftware serial test started");
17.
18. }
19.
20. void loop() {
21.
22. swSerial.print("Hello: " + String(i));
23. i++;
24.
25. while (swSerial.available() > 0)
26. {
27. char c = swSerial.read(); //gets one byte from serial buffer
28. DataString += c; //makes the string DataString
29.
30. }
31.
32. if (DataString != "")
33. {
34. Serial.println(DataString);
35. }
36.
37.
38. DataString = "";
39. delay(1000);
40.
41.
42. }

```

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



มาต่อ กันที่ เคสของตอนนี้ของเรากันดีกว่า นั่นก็คือการ Bridge ระหว่าง Local Mesh Network ของเรา เพื่อส่งข้อมูลออกไปยังโลกภายนอกกัน โดยที่ Scenario ที่เราจะจำลองขึ้นมาจะใช้ Nodemcu ทั้งหมด 5 ตัวดังรูปด้านล่างนี้



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

*MQTT คืออะไร โดยละเอียดไว้ติดตามกันต่อตอนที่ 5 แต่คร่าวๆคือ messaging protocol ตัวหนึ่งซึ่งใช้รับส่งข้อความผ่าน IP Network ารมณ์คล้ายๆ Instant Message ซึ่งการรับส่งข้อความ จะใช้ลักษณะของการ Public(ส่ง)-Subscribe(รับ) กับ Topic ที่เราต้องการ ซึ่ง MQTT จะใช้มากในงานของ IoT เพราะไม่ซับซ้อน กิน resource น้อยและค่อนข้าง realtime

ในส่วนของ Code ก็ประกอบด้วย 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

```

1. #include "painlessMesh.h"
2. #include "DHT.h"
3.
4. #define DHTPIN D4
5. #define DHTTYPE DHT22
6.
7. #define MESH_PREFIX      "HelloMyMesh"
8. #define MESH_PASSWORD    "hellomymeshnetwork"
9. #define MESH_PORT        5555
10.
11. DHT dht(DHTPIN, DHTTYPE);
12.
13. void receivedCallback( uint32_t from, String &msg );
14.
15. painlessMesh mesh;
16.
17. size_t logServerId = 0;
18.
19. // Send message to the logServer every 5 seconds
20. Task myLoggingTask(5000, TASK_FOREVER, []() {
21.
22.     float h = dht.readHumidity();
23.     float t = dht.readTemperature();
24.
25.
26.     DynamicJsonBuffer jsonBuffer;
27.     JsonObject& msg = jsonBuffer.createObject();
28.     msg["nodename"] = "mcu-t3"; //change for identify for the node that send data
mcu-tl to mcu-t3
29.     msg["NodeID"] = mesh.getNodeId();
30.     msg["Temp"] = String(t)+"C";
31.     msg["Humidity"] = String(h)+"%";
32.
33.     String str;
34.     msg.printTo(str);
35.     if (logServerId == 0) // If we don't know the logServer yet
36.         mesh.sendBroadcast(str);
37.     else
38.         mesh.sendSingle(logServerId, str);
39.
40.     // log to serial
41.     msg.printTo(Serial);
42.     Serial.printf("\n");
43. });
44.
45. void setup() {
46.
47.     Serial.begin(115200);
48.     Serial.println("Begin DHT22 Mesh Network test!");
49.
50.     dht.begin();
51.
52.     mesh.setDebugMsgTypes( ERROR | STARTUP | CONNECTION ); // set before init() so
that you can see startup messages
53.
54.     mesh.init( MESH_PREFIX, MESH_PASSWORD, MESH_PORT, STA_AP, AUTH_WPA2_PSK, 6 );
55.     mesh.onReceive(&receivedCallback);
56.
57.     // Add the task to the mesh scheduler
58.     mesh.scheduler.addTask(myLoggingTask);
59.     myLoggingTask.enable();
60.
61. }
62.
63. void loop() {
64.     // put your main code here, to run repeatedly:
65.
66.     mesh.update();
67.
68. }
69.
70.
71. void receivedCallback( uint32_t from, String &msg ) {
72.     Serial.printf("logClient: Received from %u msg=%s\n", from, msg.c_str());
73.
74.     // Saving logServer
75.     DynamicJsonBuffer jsonBuffer;
76.     JsonObject& root = jsonBuffer.parseObject(msg);
77.     if (root.containsKey("topic")) {
78.         if (String("logServer").equals(root["topic"].as<String>())) {
79.             // check for on: true or false
80.             logServerId = root["nodeId"];
81.             Serial.printf("logServer detected!!!\n");
82.         }
83.         Serial.printf("Handled from %u msg=%s\n", from, msg.c_str());
84.     }
85. }

```

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

Code ที่ใช้สำหรับ Server Node

```

1. #include "painlessMesh.h"
2. #include <SoftwareSerial.h>
3.
4. #define MESH_PREFIX      "HelloMyMesh"
5. #define MESH_PASSWORD    "hellomymeshnetwork"
6. #define MESH_PORT        5555
7.
8. SoftwareSerial swSerial(D1, D2, false, 128); //Define hardware connections RX, TX
9.
10. painlessMesh mesh;
11.
12. // Send my ID every 10 seconds to inform others
13. Task logServerTask(10000, TASK_FOREVER, []() {
14.     DynamicJsonBuffer jsonBuffer;
15.     JsonObject& msg = jsonBuffer.createObject();
16.     msg["topic"] = "logServer";
17.     msg["nodeId"] = mesh.getNodeId();
18.
19.     String str;
20.     msg.printTo(str);
21.     mesh.sendBroadcast(str);
22.
23.     // log to serial
24.     msg.printTo(Serial);
25.     Serial.printf("\n");
26. });
27.
28. void setup() {
29.
30.     pinMode(D1, INPUT);
31.     pinMode(D2, OUTPUT);
32.
33.     Serial.begin(115200); //Initialize hardware serial with baudrate of 115200
34.     swSerial.begin(115200); //Initialize software serial with baudrate of 115200
35.
36.
37.     mesh.setDebugMsgTypes( ERROR | CONNECTION | S_TIME );
38.
39.     mesh.init( MESH_PREFIX, MESH_PASSWORD, MESH_PORT, STA_AP, AUTH_WPA2_PSK, 6 );
40.     mesh.onReceive(&receivedCallback);
41.
42.     mesh.onNewConnection([](size_t nodeId) {
43.         Serial.printf("New Connection %u\n", nodeId);
44.     });
45.
46.     mesh.onDroppedConnection([](size_t nodeId) {
47.         Serial.printf("Dropped Connection %u\n", nodeId);
48.     });
49.
50.     // Add the task to the mesh scheduler
51.     mesh.scheduler.addTask(logServerTask);
52.     logServerTask.enable();
53. }
54.
55. void loop() {
56.     mesh.update();
57. }
58.
59. void receivedCallback( uint32_t from, String &msg ) {
60.     Serial.printf("logServer: Received from %u msg=%s\n", from, msg.c_str());
61.
62.     swSerial.print(msg.c_str()); // Print received data in Mesh Network to Software
63.     Serial Port
64. }
```

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 หรืออะไรก็ตามใช้หลักการเดียวกันเลย

Code ที่ใช้สำหรับ Gateway Node เพื่อส่งข้อมูลไปยัง MQTT server ซึ่งถ้าใครอยากระบดลองในส่วนนี้ ก็ติดตามจากตอนที่ 5 ในเรื่องของการตั้ง MQTT Server บน Google Cloud หรืออาจจะหาใช้บริการ Cloud MQTT ซึ่งมีส่วนให้บริการฟรีก็ได้ครับ

```

1. #include <SoftwareSerial.h>
2. #include <ESP8266WiFi.h>
3. #include <PubSubClient.h>
4.
5. const char* ssid = "xxx"; //wifi ssid
6. const char* password = "xxx"; //wifi passwd
7.
8. IPAddress mqtt_server(xx, xx, xx, xx); //ip of mqtt server
9.
10. WiFiClient espClient;
11.
12. long lastMsg = 0;
13. char msg[100];
14. int value = 0;
15.
16. String DataString;
17.
18.
19. SoftwareSerial swSerial(D1, D2, false, 256); //Define hardware connections RX, TX
20.
21.
22. void callback(char* topic, byte* payload, unsigned int length) {
23.   Serial.print("Message arrived [");
24.   Serial.print(topic);
25.   Serial.print("] ");
26.   for (int i = 0; i < length; i++) {
27.     Serial.print((char)payload[i]);
28.   }
29.   Serial.println();
30. }
31.
32.
33. PubSubClient client(mqtt_server, 1883, callback, espClient);
34.
35. void setup() {
36.
37.   pinMode(D1, INPUT);
38.   pinMode(D2, OUTPUT);
39.
40.   Serial.begin(115200); //Initialize hardware serial with baudrate of 115200
41.   swSerial.begin(115200); //Initialize software serial with baudrate of 115200
42.
43.   Serial.println("Bridget Node Gateway to MQTT");
44.
45.   Serial.println();
46.   Serial.print("connecting to ");
47.   Serial.println(ssid);
48.   WiFi.begin(ssid, password);
49.   while (WiFi.status() != WL_CONNECTED) {
50.     delay(1000);
51.     Serial.print(".");
52.   }
53.   Serial.println("");
54.   Serial.println("WiFi connected");
55.   Serial.println("IP address: ");
56.   Serial.println(WiFi.localIP());

```

```

57.     client.connect("MeshGateway", "xxx", "xxx"); //MeshGateway is node name, change
58.     xxx to user/passs of your server
59.     client.setCallback(callback);
60.     client.subscribe("command");
61.
62. }
63.
64.
65. void loop() {
66.
67.
68.     while (swSerial.available() > 0)
69.     {
70.         char c = swSerial.read(); //gets one byte from serial buffer
71.         DataString += c; //makes the string DataString
72.     }
73.
74.
75.     if (DataString != "")
76.     {
77.         Serial.println(DataString);
78.
79.         if (!client.connected()) {
80.             reconnect();
81.         }
82.
83.         client.loop();
84.
85.         DataString.toCharArray(msg, 100);
86.         client.publish("envMesh", msg); //publish message to mqtt server with topic
envMesh
87.
88.     }
89.
90.
91.
92.     DataString = "";
93.     delay(1000);
94.
95.
96. }
97.
98. void reconnect() {
99.     // Loop until we're reconnected
100.    while (!client.connected()) {
101.        Serial.print("Attempting MQTT connection...");
102.        // Attempt to connect
103.        if (client.connect("MeshGateway", "xxx", "xxx")) { //username of your mqtt
server
104.            Serial.println("connected");
105.            // Once connected, publish an announcement...
106.            client.publish("outTopic", "hello world");
107.            // ... and resubscribe
108.            client.subscribe("inTopic");
109.        } else {
110.            Serial.print("failed, rc=");
111.            Serial.print(client.state());
112.            Serial.println(" try again in 5 seconds");
113.            // Wait 5 seconds before retrying
114.            delay(10000);
115.        }
116.    }
117. }

```

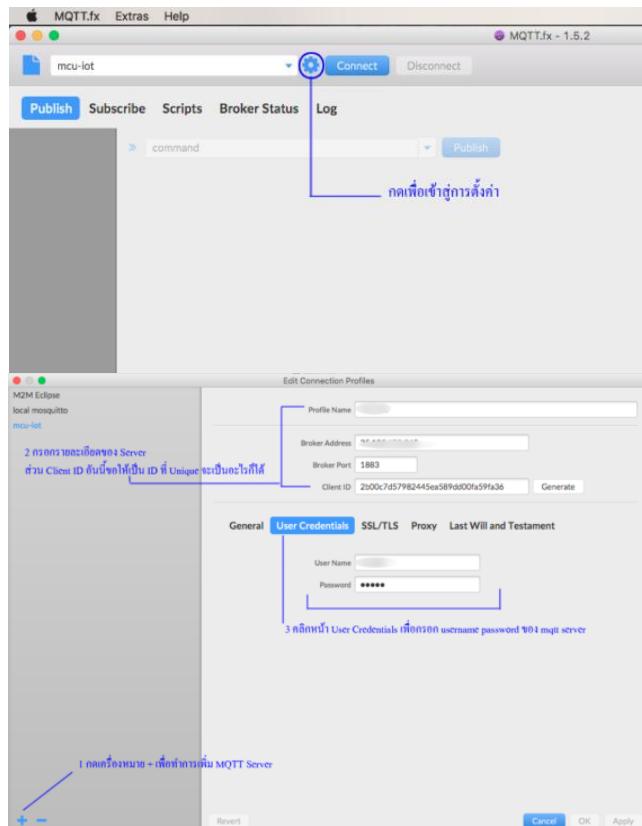
เมื่อเปิด Nodemcu ทั้ง 5 ตัวขึ้นมา เหตุการณ์จากตอนที่ 2 ก็จะปรากฏขึ้นดัง step ต่อไปนี้

- mcu-t1 ถึง mcu-t3 จะ join เข้า mesh network และส่งค่า broadcast ไปยังทุก Node และคอยฟังว่าใน Mesh Network นี้มี Server Node มั้ย ถ้ามีก็จะทำการบันทึก Server NodeID และหลังจากนั้นก็ส่งเป็นแบบระบุปลายทางไปที่ Server Node ละ
- Step นี้ที่จะเพิ่เข้ามาก็คือ ที่ Server Node เมื่อได้รับค่าที่ส่งมาภายใน Mesh Network ก็จะทำการ Forward ต่อไปยัง Gateway Node ผ่านทาง software serial port ได้อะไรมาก็ส่งไปอย่างนั้น ให้

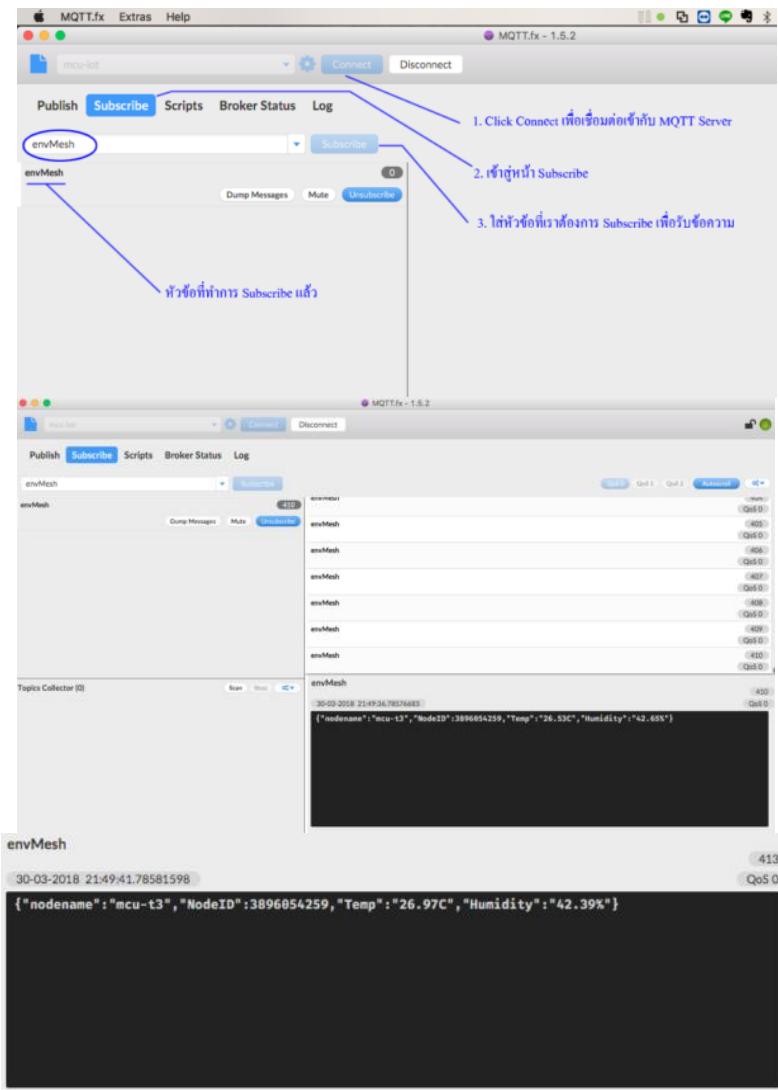
- ดู code ในส่วนของ receivedCallback ที่จะทำการ print ค่าอุณหภูมิ software serial
- ```
swSerial.print(msg.c_str());
```
- ที่ Gateway Node เมื่อได้รับค่าที่ส่งมาจาก software serial port ก็จะทำการส่งต่อค่านี้ไปยัง mqtt server

```
/dev/cu.SLAB_USBtoUART
Send
{
 "nodename": "mcu-t1", "NodeID": 3896718322, "Temp": "25.90C", "Humidity": "43.80%"
 ...
 "nodename": "mcu-t2", "NodeID": 3896712281, "Temp": "25.70C", "Humidity": "42.50%"
 ...
 "nodename": "mcu-t3", "NodeID": 3896718322, "Temp": "25.90C", "Humidity": "43.80%"
 ...
 "nodename": "mcu-t1", "NodeID": 3896718322, "Temp": "26.79C", "Humidity": "43.46%"
 ...
 "nodename": "mcu-t2", "NodeID": 3896712281, "Temp": "25.70C", "Humidity": "42.50%"
 ...
 "nodename": "mcu-t1", "NodeID": 3896718322, "Temp": "25.90C", "Humidity": "43.80%"
 ...
 "nodename": "mcu-t3", "NodeID": 3896718322, "Temp": "26.66C", "Humidity": "42.13%"
 ...
 "nodename": "mcu-t2", "NodeID": 3896712281, "Temp": "25.70C", "Humidity": "42.50%"
 ...
 "nodename": "mcu-t1", "NodeID": 3896718322, "Temp": "25.90C", "Humidity": "43.80%"
 ...
 "nodename": "mcu-t3", "NodeID": 3896718322, "Temp": "26.93C", "Humidity": "42.85%"
 ...
 "nodename": "mcu-t2", "NodeID": 3896712281, "Temp": "25.70C", "Humidity": "42.50%"
 ...
 "nodename": "mcu-t1", "NodeID": 3896718322, "Temp": "25.90C", "Humidity": "43.80%"
 ...
 "nodename": "mcu-t3", "NodeID": 3896718322, "Temp": "26.78C", "Humidity": "41.91%"
 ...
 "nodename": "mcu-t2", "NodeID": 3896712281, "Temp": "25.70C", "Humidity": "42.50%"
 ...
 "nodename": "mcu-t1", "NodeID": 3896718322, "Temp": "25.90C", "Humidity": "43.80%"
 ...
 "nodename": "mcu-t3", "NodeID": 3896718322, "Temp": "26.55C", "Humidity": "42.99%"
 ...
 "nodename": "mcu-t2", "NodeID": 3896712281, "Temp": "25.70C", "Humidity": "42.50%"
 ...
 "nodename": "mcu-t1", "NodeID": 3896718322, "Temp": "25.90C", "Humidity": "43.80%"
 ...
 "nodename": "mcu-t3", "NodeID": 3896718322, "Temp": "26.87C", "Humidity": "42.47%"
 ...
 "nodename": "mcu-t2", "NodeID": 3896712281, "Temp": "25.70C", "Humidity": "42.50%"
 ...
 "nodename": "mcu-t1", "NodeID": 3896718322, "Temp": "25.90C", "Humidity": "43.90%"
 ...
 "nodename": "mcu-t3", "NodeID": 3896718322, "Temp": "26.86C", "Humidity": "42.07%"
 ...
 "nodename": "mcu-t2", "NodeID": 3896712281, "Temp": "25.70C", "Humidity": "42.50%"
 ...
}
Autoscroll Both NL & CR 115200 baud Clear output
```

ส่วนผลลัพธ์ที่อ่านค่าได้จาก MQTT Server นั้นผมใช้โปรแกรม MQTT.fx ซึ่งเป็น Client ที่มีให้ใช้งานได้หลากหลาย platform เลยเนื่องจากว่าเป็น Java based ก็สามารถโหลดได้จากเวปของ MQTT.fx ได้เลยครับ การใช้งานก็เริ่มจากการตั้งค่ากันก่อน ที่หน้าแรกให้คลิกที่รูปเพื่อง เพื่อเข้าสู่หน้าของการตั้งค่า แล้วก็ตั้งค่าตาม MQTT Server ของแต่ละท่านที่ใช้กันอยู่เลยครับ หรือจะใช้บริการฟรีของ Cloud MQTT ก็ได้



ทำการ Subscribe หัวข้อ envMesh ตาม Code ที่อยู่ใน Gateway Node



จากรูป ก็จะเห็นข้อความที่อ่านได้จาก MQTT Server ก็จะเป็นรูปแบบ JSON โดยส่งมาเป็นทอดๆ จาก Local Mesh Network → Server Node ภายใน Mesh Network → Gateway Node ผ่านการ Bridge โดยใช้ Software Serial จากนั้นก็ส่งต่อไปที่ MQTT Server ผ่าน WiFi/Internet → และเรา ก็ใช้โปรแกรม MQTT.fx เข้าไป Subscribe เพื่ออ่านค่าที่ส่งมาให้หัวข้อ envMesh

จริงๆ สำหรับวันนี้ ใจความหลักเลยจะเป็นเรื่องของการใช้งาน Software Serial ที่ใช้ในการสื่อสารกันระหว่าง Nodemcu 2 ตัว ส่วนเรื่อง MQTT Server นั้นก็เป็นการทิ้งปมเพื่อให้ไปค้นกันต่อ หรือจะรอถึงตอนที่ 5 ก็ได้กับการ Setup MQTT Server ขึ้นมาใช้งานเองบน 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/>

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

บทความในตอนนี้ก็จะใช้เวลาเขียนนานหน่อย เนื่องด้วยงานที่ทำ แล้วก้ออุปกรณ์เจ้ากรรม Heltec ESP32+OLED 915MHz ที่สั่งมาดันเสียไปทั้งคู่ ก็เลยเว้นช่วงไว้รออุปกรณ์มาให้ครบเพื่อทดสอบ ซึ่งจากประสบการณ์ที่สั่งของจากจีนมาทดลอง กรณีที่ไม่ได้สั่งแบบนำเข้ามาทดลองแบบเป็นทางการ ก็ควรสั่งเข้ามาอย่างน้อย 2-3 ตัวด้วยกัน เพื่อตัวหนึ่งใช้งานไม่ได้จะได้ทดสอบอีกตัวหนึ่ง จะได้รู้ว่าอะไรมันเป็นที่ code ของเราหรือว่า เป็นที่อุปกรณ์กันแน่ ซึ่งสุดท้ายตัวนึง OLED เสีย อีกตัวนึง LoRa Module เสีย จบกันกับย่าน 915MHz



เนื้อหาหลักๆในตอนนี้ก็จะเกี่ยวกับเรื่องของ LPWAN (Low Power Wide Area Network) อย่าง LoRa ทั้งการใช้งาน ข้อดีข้อเสีย กฎหมายที่เกี่ยวข้องในการนำมาใช้งาน เพราะถ้าติดตามอ่านมาตั้งแต่ตอนที่ 1 (ESP8266/ESP32: Introduction & Painlessmesh) ใจความหลักในการใช้งาน Mesh Network ด้วย Painlessmesh นั้นจะไปตั้งแต่ตอนที่ 3 แล้วครับ ที่เหลือเป็นการประยุกต์มากกว่า ว่าเราจะใช้งาน Mesh Network ให้เหมาะสมยังไง หรือเพื่อจุดประสงค์ไหน เช่น

- เพื่อลด Single Point of Failure
- เพื่อขยายพื้นที่การครอบคลุมในการส่งข้อมูล
- ลดต้นทุนในการขยายจุดกระจายสัญญาณ

จากนั้นการส่งข้อมูลจาก Mesh Network ซึ่งเปรียบเหมือน Local Network ของเรา เราจะ Bridge ยังไงเพื่อให้ข้อมูลเหล่านั้นไปยัง Server, Database, Cloud, MQTT Broker หรืออะไรก็แล้วแต่ที่อยู่ Network อีก晚หนึ่งหรือผ่าน Internet ซึ่งก็จะได้ทดลองทำกันไปในตอนที่ 3 (ESP8266/ESP32: Painlessmesh Bridge) ซึ่งถ้าใครจะไม่ใช้ Software Serial ในการ Bridge ก็ยังสามารถทำได้อีกหลายวิธี ซึ่งวันนี้เราจะมา Bridge ผ่าน LoRa กัน

## LoRa vs LoRaWan

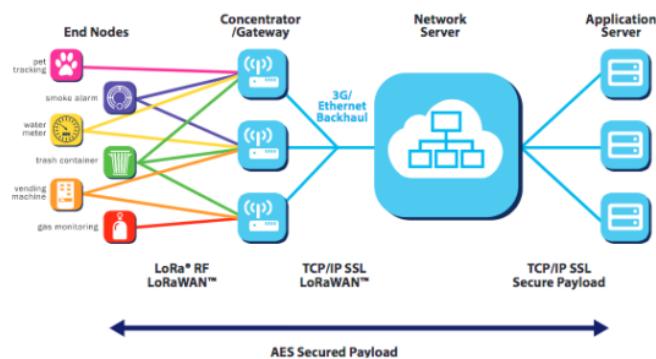
ผู้ให้บริการทางด้านโทรคมนาคมในบ้านเราว่าช่วงนี้ ทั้ง AIS (AIS NB-IoT) และ True (True IoT) ก็เปิดตัวบริการ NB-IoT อกมา ส่วน CAT ก็มาทางสาย LoRa (LoRa IoT by CAT) ซึ่งทั้งคู่ก็อกมาเพื่อรับการสื่อสารสำหรับบริการประเภท IoT ในส่วนของ AIS และ True ที่เป็นผู้ให้บริการมือถืออยู่แล้ว มีความถี่ที่ใช้ในการจัดสรรสำหรับบริการ LTE นำมาให้บริการ NB-IoT ก็จะประหยัดต้นทุนไปได้เยอะแ俣เป็นการ Utilize ความถี่ที่ตัวเองได้รับอนุญาตมาด้วย เท่าที่เห็นก็เป็น 3 ผู้บริการหลักๆที่เปิดตัวอกมา ซึ่งหัวข้อของเราในวันนี้ก็จะคุยกันในเรื่องของ LoRa และการใช้งานร่วมกัน Mesh Network จะนั่นหมายความว่าแตกต่างกันระหว่าง LoRa และ LoRaWan กันก่อน



LoRa จะเป็นการสื่อสารในระดับ Physical Layer ซึ่งหมายความว่าจะใช้งานในลักษณะแบบ P2P ไม่มี Encryption ส่งข้อมูลกันแบบ Broadcast ออกไปเลย ฉะนั้นถ้ามีคุณตั้ง LoRa Node ขึ้นมาแล้วใช้ค่า Setting เดียวกับเรา ก็สามารถที่จะเห็นข้อมูลได้ ซึ่ง LoRa ถูกพัฒนาเป็น Wireless Data Communication โดยบริษัท Cycleo (Grenoble, France) จากนั้นผู้ผลิตชิป Semtech ก็เข้าไปซื้อบริษัทนี้มาในปี 2012 ถ้าดูตามสเปคของชิปยอดฮิตอย่าง SX1276 ที่รองรับย่านความถี่ 920-925MHz ตามประกาศของ กสทช ก็จะเห็นว่าที่ LoRa นั้นส่งได้ไกลนั้นส่วนนึงก็ เพราะ High sensitivity ที่รองรับถึง -148dBm เลย (ข้อมูล Semtech SX1276) แต่ใช้งานจริงจะได้เท่าไหร่ต้องดูกัน



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



ฉะนั้นการจะเลือกใช้ LoRa หรือ LoRaWan ก็ควรเลือกให้เหมาะสมกับ Application ที่จะใช้งานด้วย และที่สำคัญควรอยู่ตามข้อกำหนดของ กสทช ด้วยครับเนื่องจากคลื่นความถี่เป็นทรัพยากรส่วนรวมที่มีการใช้งานร่วมกัน ถ้าใช้ LoRa ก็ต้องดูในเรื่องของ Duty Cycle เองด้วยไม่ใช่ส่งข้อมูลตลอดเวลา จองการใช้งานคลื่นความถี่

ตลอด รวมถึงกำลังส่งที่ถ้ามากไป ก็จะไปรบกวนการใช้งานของอุปกรณ์หรือ Application ที่ใช้งานในความถี่เดียวกัน

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

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

### **Limitations**

เรา마다ข้อดีข้อเสียของการใช้ LoRaWan กันบ้าง ไม่ใช่ว่าเทคโนโลยีที่ดีอย่าง LoRa/LoRaWan จะเหมาะสมครอบจักรวาลสำหรับทุก Application ฉันได้จัดนั้น Mesh Network ก็จะเหมาะสมกับ Application ในบางประเภท หรือบางงาน เพราะถ้าเราต้องการเอา LoRa/LoRaWan ไปส่งข้อมูลแบบ Realtime ก็คงไม่เหมาะสมนั่นเองจะกล่าวเป็นการจองช่องสัญญาตตลอดเวลา หรือให้ส่งข้อมูลขนาดใหญ่ๆ ก็คงไม่เหมาะสมเช่นกัน 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**

เนื่องจากคลื่นความถี่ที่เป็นทรัพยากรที่ต้องถูกจัดสรรเพราะมีอยู่จำกัด และต้องมีกำกับดูแลในเรื่องของการใช้งาน ในประเทศไทยจะมีองค์กรที่ทันสมัยมากๆ องค์กรหนึ่งชื่อว่า กสทช (สำนักงานคณะกรรมการ

กิจการกระจายเสียง กิจการโทรทัศน์ และกิจการโทรคมนาคมแห่งชาติ) ที่เดียวๆ ก็จะขอบนีคโนอุกมาให้เข้าไว้ว่า เด่วย์ดใบอนุญาตมั่ง ปรับบูรณาี้นั่นนั่ง แล้วสุดท้ายเรื่องก์เงียบไป มาดูเรื่องการใช้งานความถี่ของเรากันดีกว่าครับ

จากข้อมูลที่agmaจากเวปของหน่วยงาน กสทช เองบอกได้เลยว่าเล่นอาจงที่เดียว บางที่ค้นหาด้วยเลข ความถี่ต้องใช้เลขไทย ไม่เงินไม่เจอ ซึ่งผมรวมมาได้ดังนี้โดยอิงตามความถี่ย่านที่เรามักจะเห็นคุ้นเคยเห็นการใช้งานกันสำหรับ LoRa, RFID และ RC นั่นก็คือ EU433 ISM Band, AS923-925MHz

- ตารางกำหนดคลื่นความถี่แห่งชาติ 2560 (Download Link)
- กฎาคม 2560: เอกสารประกอบการรับฟังความคิดเห็นสาธารณะ ร่างประกาศเกี่ยวกับการใช้คลื่นความถี่ 920-925MHz (Download Link)
- สิงหาคม 2560: ร่างประกาศเกี่ยวกับการใช้คลื่นความถี่ 920-925MHz และแนวทางการอนุญาตเพื่อประกอบกิจการ IoT (Download Link)
- พฤษจิกายน 2560: ราชกิจจานุเบกษา – หลักเกณฑ์การอนุญาตให้ใช้คลื่นความถี่ย่าน 920-925MHz (Download Link)
- พฤษจิกายน 2560: ราชกิจจานุเบกษา – มาตรฐานทางเทคนิคของเครื่องโทรคมนาคมและอุปกรณ์สำหรับเครื่องวิทยุคมนาคมที่ไม่ใช่ประเภท RFID ซึ่งใช้คลื่นความถี่ย่าน 920-925MHz (Download Link)
- มกราคม 2561: ราชกิจจานุเบกษา – หลักเกณฑ์และเงื่อนไขการอนุญาตให้ใช้คลื่นความถี่สำหรับอากาศยานซึ่งไม่มีนักบินสำหรับใช้งานเป็นการทั่วไป (Download Link)
- กุมภาพันธ์ 2561 ราชกิจจานุเบกษา – เครื่องวิทยุคมนาคมและสถานีวิทยุคมนาคมที่ได้รับการยกเว้นไม่ต้องได้รับใบอนุญาตตามพระราชบัญญัติวิทยุคมนาคม (Download Link)
- ความถี่วิทยุและใบอนุญาตวิทยุโทรคมนาคมประเภท RFID (Download Link)
- การปรับปรุงกฎระเบียบด้านการบริหารคลื่นความถี่และทรัพยากรโทรคมนาคมเพื่อรองรับการพัฒนาของ Internet of Things ในประเทศไทย (Download Link)

ฉะนั้นสำหรับส่วนของคลื่นความถี่ 433MHz ซึ่งที่เราจะทดสอบกันสำหรับ LoRa นั้น ก็จะไม่ได้เกี่ยวกับการใช้งานทั้ง RFID หรือการใช้งานในด้าน RC กับอากาศยานไร้คนขับ แต่จะทดสอบโดยอาศัยความตามเครื่องวิทยุคมนาคมและสถานีวิทยุคมนาคมที่ได้รับการยกเว้นไม่ต้องได้รับใบอนุญาต ซึ่งย่าน 433MHz นั้นห้ามส่งเกิน 10 มิลลิวัตต์

สำหรับย่าน 920-925MHz นั้นซึ่งมีข้อกำหนดในส่วนของ มาตรฐานทางเทคนิคของเครื่องโทรคมนาคม และอุปกรณ์ กสทช.มท. 1033 – 2560 ว่าด้วยเรื่องเครื่องวิทยุคมนาคมที่ไม่ใช่ประเภท Radio Frequency Identification: RFID ซึ่งใช้คลื่นความถี่ย่าน 920-925MHz เกาะเอิร์ตซ์ กกำหนดไว้ว่า ไม่เกิน 50 มิลลิวัตต์ได้รับยกเว้นไม่

ต้องได้รับใบอนุญาตให้ทำมีใช้นำเข้าและนำออกซึ่งเครื่องวิทยุคมนาคมและใบอนุญาตให้ตั้งสถานีวิทยุคมนาคมแต่ไม่ได้รับยกเว้นใบอนุญาตให้ค้าซึ่งเครื่องวิทยุคมนาคม

## 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)

ถ้ามีดังนี้ตาม Frequency Plan และตามช่วงความถี่ที่ กสทช ปลดล็อกให้กับอุปกรณ์และบริการ IoT ก็สามารถใช้ได้สองช่วงเลยครับ ทั้ง AS920 กับ AS923 (The Things Network Frequency Plan)

จากข้อมูลด้านบนเนี่ย ก็ต้องอ่านกันดีๆ หลักๆ ก็คือรยิดตามคลื่นความถี่ที่ใช้งานและจุดประสงค์ที่ใช้งานจากนั้นก็ต้องตามหลักเกณฑ์และเงื่อนไขที่ประกาศในราชกิจจานุเบกษา และก็ยังต้องดูด้วยว่าในการประกาศนั้นมีการยกเลิกประกาศตัวเก่าไว้หรือไม่ หรือพวกระยะห่างที่ว่าเว้นแต่กำหนดเป็นอย่างอื่น พวกร่างกับการรับฟังความคิดเห็นสาธารณะก็อาจใช้ประกอบได้ กรณีที่สุดหากมีข้อสงสัยในเรื่องของการใช้งานความถี่ การผลิตหรือนำเข้า เรื่องอุปกรณ์คมนาคมที่ใช้ความถี่ ก็ควรขอคำตอบจาก กสทช เป็นหลักซึ่งก็ควรทำเป็นหนังสือแจ้งไปเพื่อให้ กสทช ตอบกลับว่าสามารถทำได้ หรือไม่สามารถทำได้ หรือให้ยึดตามข้อกำหนดใหม่

## Scenario

กรณีไหนบ้างที่เราควรจะ Bridge ผ่าน LoRa ซึ่งข้อดีของ LoRa เลยก็คือเรื่องของ Power Consumption ที่ต่ำ และระยะในการส่งที่สูง ฉะนั้นด้วยคุณสมบัตินี้ แทนที่จะต้องการเก็บข้อมูล หรือควบคุมอุปกรณ์ จะต้องวาง Mesh Network ให้ครอบคลุมมาถึงจุดที่จะส่งข้อมูลออก Internet อาจจะไม่จำเป็น แต่อาศัยการส่งระยะไกล ผ่าน LoRa มาแล้วค่อยมาออก Internet จากฝั่งของ LoRa Node ที่ทำตัวเป็น Gateway ก็ได้ สำหรับตอนที่ 4 นี้ ก็จะเป็นการใช้งานผ่าน LoRa นะครับ แต่ถ้าใครอยากรู้ Advance Setup LoRaWan Gateway แล้วใช้งานผ่าน LoRaWan ก็ได้ อย่างที่บอกไว้ครับ หลักการเดียวกันกับตอนที่ 3 ลองมาคิดกันเล่นๆครับ

### Scenario 1: Smart Farm

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

เคล็ดลับการใช้ Mesh Network เหมาะมาก เพราะได้ประโยชน์ในเรื่องของการครอบคลุมพื้นที่ที่กว้าง จากการกระจายตัวของ Node แล้วสร้างเป็น Mesh Network โดยที่จุดเชื่อมต่อสุดท้ายอาจอยู่ที่ชายขอบ หรือศูนย์กลางของ Mesh Network ก็จะเหมาะสมมาก สามารถลดต้นทุนในเรื่องของการสร้าง Network สำหรับการสื่อสารของแต่ละ Node ได้

### Scenario 2: Falling Rock Alarm

สำหรับเคล็ดลับนี้เป็นเคล็ดลับที่ได้หัวน ที่ใช้ในการตรวจสอบหินที่滾จากภูเขา เนื่องจากได้หัวนนั้นมีสภาพภูมิประเทศเป็นภาษา และอยู่ในเขตแผ่นดินไหว ซึ่งถนนบางเส้นนั้นก็ตัดผ่านภูเขามาเนื่องจากภูมิประเทศ ไทย ซึ่งบางครั้งก็จะมีหินร่วงหล่นลงมา กว่าจะทราบเหตุก็ต้องมีคนโทรแจ้งจากที่ไปเจอ



ซึ่งที่ได้หัวนได้มีการติดตั้งเซนเซอร์ตรวจจับการสั่นสะเทือนของแนวตะข่ายกันหินตามถนนไว้ เพื่อตรวจจับว่ามีหินร่วงหล่นลงมา หรือมีดินสไลเดิลลงมาปิดถนนมั้ยโดยการใช้การสื่อสารแบบ LoRa เพราะการตรวจจับไม่ได้ตรวจตลอดทั้งแนว เนพาะบางช่วงที่ตัดผ่านภูเขามา และต้องการส่งข้อมูลระยะไกล LoRa จึงตอบโจทย์ ดี ฉะนั้นเคล็ดลักษณะแบบนี้ เราสามารถเอา Mesh Network มาทำงานร่วมกับ LoRa ได้ โดยที่ Sensor Node สามารถติดเป็นบริเวณที่กว้างขึ้นได้โดยมีต้นทุนที่ต่ำ และส่งข้อมูลแจ้งเตือนไปที่ศูนย์แจ้งเตือนด้วยการ Bridge ผ่าน LoRa ที่อาจอยู่ไกลออกไปหลายกิโล ถ้าเมืองไทยใครมีโครงการทำก็ตีมากเลยครับ ปีที่แล้วขับรถไปดอยตุงที่

เชิงรายช่วงหน้า Fen ก็จะอ่านปิตไปครึ่งเล่นเป็นบางช่วงอยู่เหมือนกัน ถ้าเราต้องการให้หน้าก็จะสามารถวางแผนการเดินทางหรือหลีกเลี่ยงเส้นทางได้ถูก

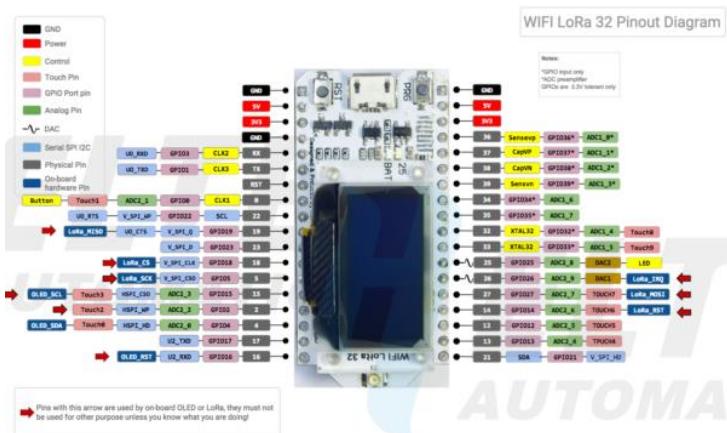
### Scenario 3: Smart Bin

เคสนี้เป็นเคสที่เกิดขึ้นจริงอีกเช่นกัน ลองคิดดูถึงเรื่องการเก็บขยะ ซึ่งในบางหมู่บ้าน ชุมชน ที่ต้องมีรถเก็บขยะวนเข้าไปเก็บทุกวันๆ หรือบางที่วันนึงอาจจะหลายรอบก็ได้ หรือบางที่เข้าไปเก็บ ขยะในถังอาจจะไม่มีเลยก็ได้

ซึ่งเคสนี้มีคนทำเป็น Product Smart Bin ที่มีทั้งตัวบดอัดขยะเพื่อเพิ่มพื้นที่การเก็บขยะได้มากขึ้น มีตัว Monitor ระดับของขยะเพื่อส่งคำสั่งไปแจ้งเตือนที่ศูนย์ว่าให้มานำเก็บขยะได้แล้ว รวมถึงจัดเส้นทางให้รถเก็บขยะได้ด้วย ซึ่งเทคโนโลยีการสื่อสารที่ใช้โดยมากก็จะเป็น LoRa หรือไม่ก็ Cellular (2G, 3G, NB-IoT) เพราะด้วยลักษณะการตั้งวางถังขยะของหมู่บ้าน หรือชุมชนใน ตปท อาจไม่เหมาะสมกับการนำ mesh มาใช้

### Wiring & Diagram

ความจริงการใช้งานจะใช้งานผ่าน Module SX1278/1276 ของ Ai-Thinker ก็ได้แต่ก็ต้องต่อสายเข้ากับ Breadboard หรือในส่วนของ RFM95W ก็ต้องมาบัดกรีหากันอีก ฉะนั้นตอนนี้อาจง่ายสุดสำหรับผม ก็ใช้บอร์ดสำเร็จรูปที่มาพร้อม ESP32+OLED ของ Heltec/TTGO เลยดีกว่า เพราะว่าทำการ wiring ไว้บน pcb เรียบร้อยแล้ว ซึ่ง pinout ก็จะเป็นดังรูปด้านล่างนี้



จาก Espressif ผู้ผลิตชิป ESP32

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

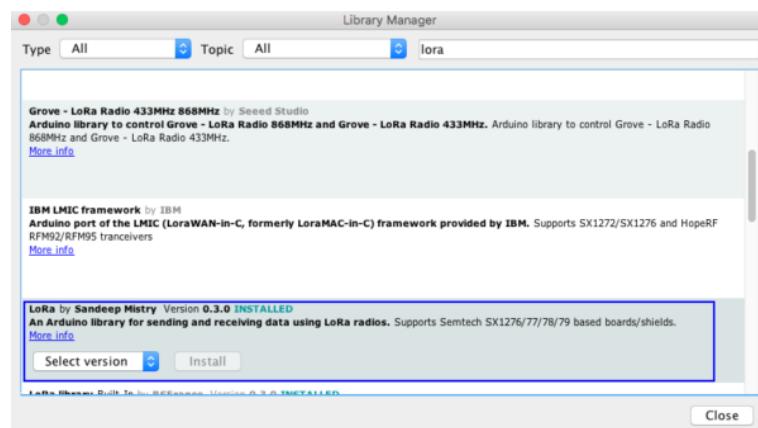
จาก Heltec ผู้ผลิตบอร์ด Heltec ESP32 LoRa

[https://github.com/Heltec-Aaron-Lee/WiFi\\_Kit\\_series#installation-instructions](https://github.com/Heltec-Aaron-Lee/WiFi_Kit_series#installation-instructions)

ข้อแตกต่างระหว่างสองบอร์ดนี้ก็คือ ถ้า Clone มาจาก Heltec จะมี Library LoRa มาด้วยซึ่งก็เป็นตัวที่ไม่มากจากของคุณ Sandeep Mistry แต่ถ้ากรณีมีการ Update ก็จะไม่ถูก Update ผ่านทาง Library manager ต้องค่อยตามจากทาง Heltec เอง ขณะเดียวกัน Clone มาจากทาง Espressif แล้วทำการติดตั้ง LoRa Library เอง ส่วนวิธีการติดตั้งก็ขึ้นกับ OS ของแต่ละท่านที่ใช้อยู่ หลังจากติดตั้งสินใจเลือกทางได้แล้วก็มาศัลย์ลงช้างบนนี้ แหล่งครับที่ตาม Instruction ในแต่ละ OS ได้เลย

LoRa Library by Sandeep Mistry

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



มาตรฐานในส่วนของ Code ที่จะใช้ทดสอบระยะกันบ้าง ตัวโปรแกรมก็จะแบ่งเป็นสองชุดคือฝั่งส่งและฝั่งรับ ฝั่งส่งก็ Count Number แล้วส่งออกไปเรื่อยๆ ส่วนฝั่งรับนั้นเนื่องด้วยไม่มี GPS Module ก็เลือกตัว GPS Stream จาก Blynk มาใช้ในการเก็บค่าพิกัดแทน เพื่อที่จะได้มารา plot ดูว่าได้ระยะเป็นอย่างไรบ้าง ซึ่งถ้าใช้บอร์ดที่ใช้ชิป SX1278 ในส่วนของ Band จะกำหนดเป็น 433.175MHz และถ้าเป็นบอร์ดที่ใช้ชิป SX1276 ในส่วนของ Band จะกำหนดเป็น 923.2MHz นอกนั้นไม่มีอะไรแตกต่าง ส่วนใครที่จะทดสอบโดยที่ไม่ได้สนใจเรื่องของ พิกัด ก็ตัด code ในส่วนของ Blynk ออกได้ครับ

## Code for Sender:

```

1. #include <SPI.h>
2. #include <LoRa.h>
3. #include "SSD1306.h"
4.
5. int counter = 0;
6.
7.
8. // GPIO5 -- SX1278's SCK
9. // GPIO19 -- SX1278's MISO
10. // GPIO27 -- SX1278's MOSI
11. // GPIO18 -- SX1278's CS
12. // GPIO14 -- SX1278's RESET
13. // GPIO26 -- SX1278's IRQ(Interrupt Request)
14.
15. //OLED pins to ESP32 0.96OLEDGPIOS via this connectin:
16. //OLED_SDA -- GPIO4
17. //OLED_SCL -- GPIO15
18. //OLED_RST -- GPIO16
19.
20. SSD1306 display(0x3c, 4, 15);
21.
22. #define SS 18
23. #define RST 14
24. #define DIO 26
25. #define BAND 433.175E6 //915E6
26.
27. void setup() {
28. Serial.begin(115200);
29.
30. pinMode(25, OUTPUT); //Send success, LED will bright 1 second
31.
32. while (!Serial);
33.
34. pinMode(16, OUTPUT);
35. digitalWrite(16, LOW); // set GPIO16 low to reset OLED
36. delay(50);
37. digitalWrite(16, HIGH); // while OLED is running, must set GPIO16 in high
38.
39. Serial.println("LoRa Sender");
40.
41. SPI.begin(5, 19, 27, 18);
42. LoRa.setPins(SS, RST, DIO);
43.
44. if (!LoRa.begin(BAND)) {
45. Serial.println("Starting LoRa failed!");
46. while (1);
47. }
48.
49. Serial.println("LoRa Initial OK!");
50.
51. // Initialising the UI will init the display too.
52. display.init();
53. display.flipScreenVertically();
54. display.setFont(ArialMT_Plain_10);
55.
56.

```

```

57. void loop() {
58. Serial.print("Sending packet: ");
59. Serial.println(counter);
60.
61. // send packet
62. LoRa.beginPacket();
63. LoRa.print("Hello ");
64. LoRa.print(counter);
65. LoRa.endPacket();
66.
67. display.clear();
68. display.setTextAlignment(TEXT_ALIGN_LEFT);
69. display.drawString(10, 5, "Sending:");
70. display.drawString(10, 20, "Hello " + String(counter));
71. // write the buffer to the display
72. display.display();
73.
74. counter++;
75. digitalWrite(25, HIGH); // turn the LED on (HIGH is the voltage level)
76. delay(1000); // wait for a second
77. digitalWrite(25, LOW); // turn the LED off by making the voltage LOW
78. delay(1000); // wait for a second
79.
80. delay(5000);
81. }
```

### Code for Receiver:

```

1. #include <SPI.h>
2. #include <LoRa.h>
3. #include "SSD1306.h"
4.
5. /* Comment this out to disable prints and save space */
6. #define BLYNK_PRINT Serial
7.
8. #include <WiFi.h>
9. #include <WiFiClient.h>
10. #include <BlynkSimpleEsp32.h>
11.
12. char auth[] = "xxxxxxxxxx"; //blynk auth code
13.
14. // Your WiFi credentials.
15. // Set password to "" for open networks.
16. char ssid[] = "xxx xxxx";
17. char pass[] = "xxxx";
18.
19. float lat;
20. float lon;
21.
22.
23.
24. // GPIO5 -- SX1278's SCK
25. // GPIO19 -- SX1278's MISO
26. // GPIO27 -- SX1278's MOSI
27. // GPIO18 -- SX1278's CS
28. // GPIO14 -- SX1278's RESET
29. // GPIO26 -- SX1278's IRQ(Interrupt Request)
30.
31. //OLED pins to ESP32 0.96OLEDGPIOS
32. //OLED_SDA -- GPIO4
33. //OLED_SCL -- GPIO15
34. //OLED_RST -- GPIO16
35.
36. SSD1306 display(0x3c, 4, 15);
37.
38. #define SS 18
39. #define RST 14
40. #define DIO 26
41. #define BAND 433.175E6 //915E6
42.
43. WidgetMap myMap(v1);
44.
45.
46. BLYNK_WRITE(V0) {
```

```

47. GpsParam gps (param);
48.
49. // Print 6 decimal places for Lat, Lon
50.
51. lat = String(gps.getLatitude(),6).toFloat();
52. lon = String(gps.getLongitude(),6).toFloat();
53.
54. Serial.print("Lat: ");
55. Serial.println(lat,7);
56.
57. Serial.print("Lon: ");
58. Serial.println(lon,7);
59.
60. Serial.println();
61.
62. }
63.
64.
65. void setup() {
66. Serial.begin(115200);
67. while (!Serial);
68.
69. pinMode(16, OUTPUT);
70. digitalWrite(16, LOW); // set GPIO16 low to reset OLED
71. delay(50);
72. digitalWrite(16, HIGH); // while OLED is running, must set GPIO16 in high
73.
74. Serial.println("LoRa Receiver");
75.
76. SPI.begin(5, 19, 27, 18);
77. LoRa.setPins(SS, RST, DIO);
78.
79. if (!LoRa.begin(BAND)) {
80. Serial.println("Starting LoRa failed!");
81. while (1);
82.
83. }
84. Serial.println("LoRa Initial OK!");
85.
86. // Initialising the UI will init the display too.
87. display.init();
88. display.flipScreenVertically();
89. display.setFont(ArialMT_Plain_10);
90.
91. Blynk.begin(auth, ssid, pass);
92.
93. myMap.clear();
94.
95. }
96.
97.
98. void loop() {
99. String tmp_string, tmp_rssi;
100.
101. // try to parse packet
102. int packetSize = LoRa.parsePacket();
103. if (packetSize) {
104. // received a packet
105. Serial.print("Received packet !");
106.
107. // read packet
108. while (LoRa.available()) {
109. //Serial.print((char)LoRa.read());
110. tmp_string += (char)LoRa.read();
111. }
112.
113. Serial.print(tmp_string);
114.
115. tmp_rssi = LoRa.packetRssi();
116.
117. // print RSSI of packet
118.
119. Serial.println(" with RSSI " + tmp_rssi);
120.
121. //Serial.println(LoRa.packetRssi());
122.
123. }
```

```

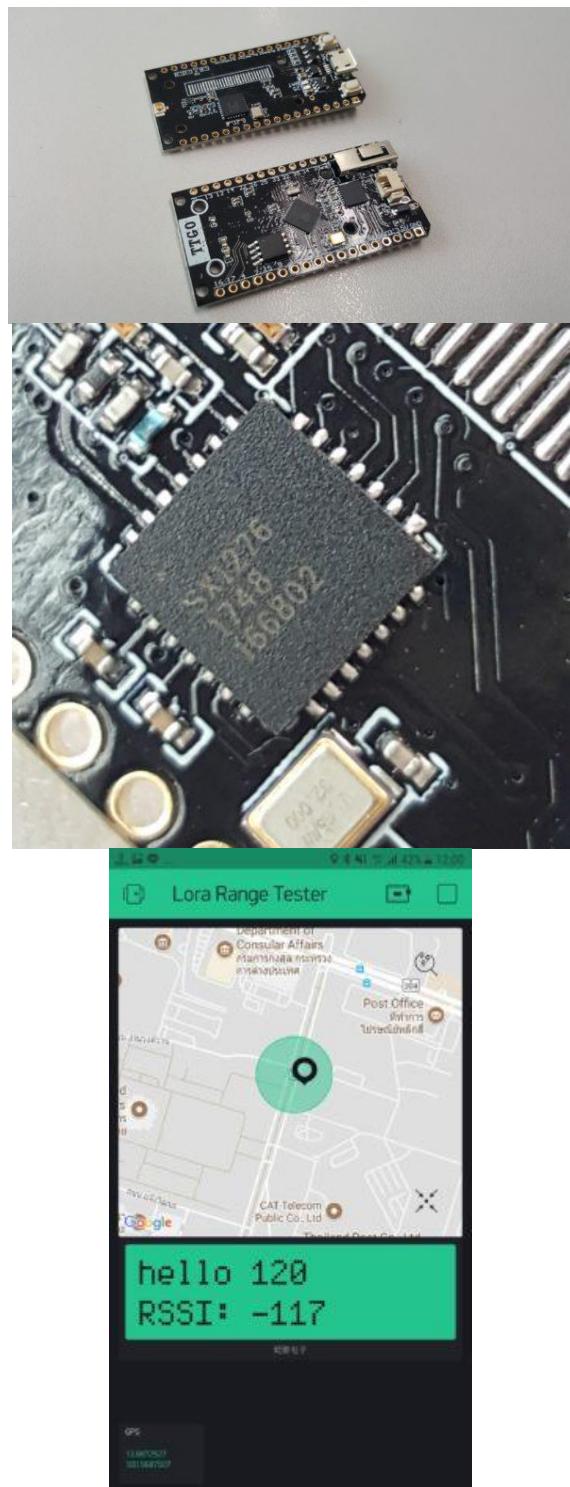
124. display.clear();
125. display.setTextAlignment(TEXT_ALIGN_LEFT);
126. display.drawString(10, 0, "Received:");
127. display.drawString(10, 15, tmp_string+" RSSI: "+tmp_rssi);
128. display.drawString(10, 30, String(lat,7));
129. display.drawString(10, 45, String(lon,7));
130. // write the buffer to the display
131. display.display();
132.
133. //BLYNK_WRITE(V0);
134. Blynk.syncVirtual(V0);
135. myMap.location(1, lat, lon, "value");
136.
137. }
138. tmp_string ="";
139. tmp_rssi = "";
140.
141. Blynk.run();
142.
143. }

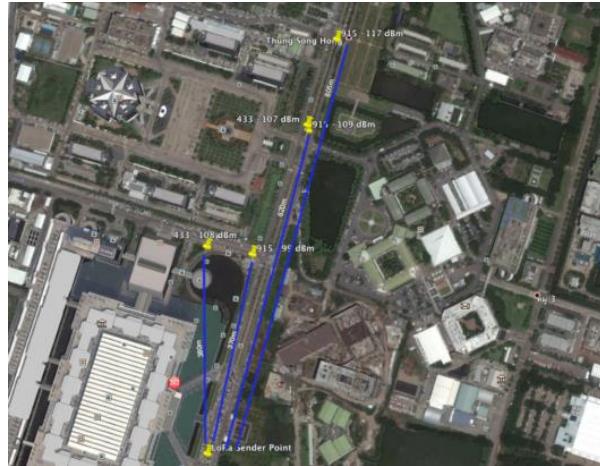
```

LoRa Range Test Result: 433.175MHz vs 923.2MHz



ในความโฉคร้ายยังพอมีโชคดีอยู่บ้าง ถึงแม้ตัว Heltec ESP32 LoRa SX1276 ที่ใช้งานย่าน 915MHz จะเสียแต่ก็ได้บอร์ดใหม่มาแทนพอให้ทดสอบ ซึ่งเป็นของ TTGO เป็น ESP32 LoRa SX1276 เมื่อันกันเพียงแต่เปลี่ยน  
มี OLED มาด้วยเท่านั้น ฉะนั้นการวัดระยะ และค่า RSSI ก็เลือกอาศัยอ่านค่า GPS จาก App Blynk บนโทรศัพท์มือถือแทน

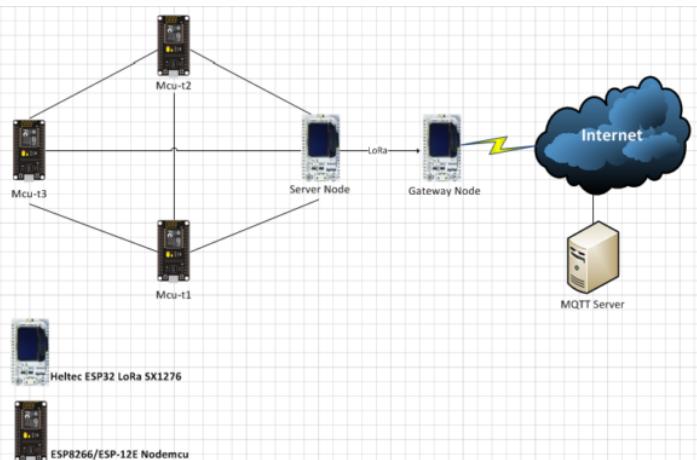




ผลการทดสอบระยะของแต่ละความถี่ที่ใช้กับตัวรับสัญญาณด้านบนเลียครับ ซึ่งถ้าเราระยะห่างผลไกลสุดซึ่งอยู่ใน Line of Sight ก็จะอยู่ที่ประมาณ 700-750 เมตรสำหรับการรับ-ส่งที่ความถี่ 923.2MHz ส่วนคลื่น 433.175MHz ระยะห่างผลก็จะอยู่ที่ประมาณ 600 เมตร ทำให้การใช้งาน LoRa นั้นค่อนข้างสมชื่อ Long Range เลย เพราะนี่คือใช้สายอากาศที่แรมมา ถ้าใช้สายอากาศดีๆ และตั้งเสาสูงน่าจะไปได้ไกลกว่านี้ แต่ทั้งนี้ทั้งนั้นกำลังส่งก็ไม่ควรเกิน กษาช กําหนดนะครับ ซึ่งเคลื่อนที่เขียนนี้ “สำหรับการทดลองเท่านั้น” เพราะจาก Code ได้มีการกำหนด TX Power ไว้ที่ 17dBm บวกกับเสาที่แรมมาน่าจะไม่เกิน +3dBm ตีกลมๆ ก็ 20dBm หรือ 100 milliwatts แล้ว (แต่ส่วนตัวคิดว่าไม่น่าจะถึง น่าจะมี loss เเยะ ถ้าอยากรู้จริงๆ คงต้องเอาอุปกรณ์มาวัด)

### Painlessmesh Bridge with Lora

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



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

```

1. สำหรับ ESP32
2. https://github.com/me-no-dev/AsyncTCP
3.
4.
5. สำหรับ ESP8266
6.
7. https://github.com/me-no-dev/ESPAsyncTCP

```

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

Code:

```

1. #include "painlessMesh.h"
2. #include "DHT.h"
3. #define DHTPIN D4
4. #define DHTTYPE DHT22
5. #define MESH_PREFIX "HelloMyMesh"
6. #define MESH_PASSWORD "hellomymeshnetwork"
7. #define MESH_PORT 5555
8. DHT dht(DHTPIN, DHTTYPE);
9.
10. void receivedCallback(uint32_t from, String &msg);
11.
12. painlessMesh mesh;
13. size_t logServerId = 0;
14. // Send message to the logServer every 10 seconds
15. Task myLoggingTask(10000, TASK_FOREVER, []() {
16.
17. float h = dht.readHumidity();
18. float t = dht.readTemperature();
19.
20. DynamicJsonBuffer jsonBuffer;
21. JsonObject& msg = jsonBuffer.createObject();
22. msg["nodename"] = "mcu-t1"; //change for identify for the node that send data
mcu-t1 to mcu-t3
23. msg["NodeID"] = mesh.getNodeId();
24. msg["Temp"] = String(t) + "C";
25. msg["Humidity"] = String(h) + "%";
26. String str;
27. msg.printTo(str);
28. if (logServerId == 0) // If we don't know the logServer yet
29. mesh.sendBroadcast(str);
30. else
31. mesh.sendSingle(logServerId, str);
32. // log to serial
33. msg.printTo(Serial);
34. Serial.printf("\n");
35. });
36.
37. void setup() {
38. Serial.begin(115200);
39. Serial.println("Begin DHT22 Mesh Network test!");
40. dht.begin();
41. mesh.setDebugMsgTypes(ERROR | STARTUP | CONNECTION); // set before init() so
that you can see startup messages
42. mesh.init(MESH_PREFIX, MESH_PASSWORD, MESH_PORT, STA_AP, AUTH_WPA2_PSK, 6);
43. mesh.onReceive(&receivedCallback);
44. // Add the task to the mesh scheduler
45. mesh.scheduler.addTask(myLoggingTask);
46. myLoggingTask.enable();

```

```

47. }
48.
49. void loop() {
50. // put your main code here, to run repeatedly:
51. mesh.update();
52. }
53.
54. void receivedCallback(uint32_t from, String &msg) {
55. Serial.printf("logClient: Received from %u msg=%s\n", from, msg.c_str());
56. // Saving logServer
57. DynamicJsonBuffer jsonBuffer;
58. JsonObject& root = jsonBuffer.parseObject(msg);
59. if (root.containsKey("topic")) {
60. if (String("logServer").equals(root["topic"].as<String>())) {
61. // check for on: true or false
62. logServerId = root["nodeId"];
63. Serial.printf("logServer detected!!!\n");
64. }
65. Serial.printf("Handled from %u msg=%s\n", from, msg.c_str());
66. }
67. }

```

## Heltec ESP32 SX1276 (Server Node + LoRa)

Code:

```

1. #include "painlessMesh.h"
2. #include <SPI.h>
3. #include <LoRa.h>
4. #include "SSD1306.h"
5.
6. // GPIO5 -- SX1278's SCK
7. // GPIO19 -- SX1278's MISO
8. // GPIO27 -- SX1278's MOSI
9. // GPIO18 -- SX1278's CS
10. // GPIO14 -- SX1278's RESET
11. // GPIO26 -- SX1278's IRQ(Interrupt Request)
12. //OLED pins to ESP32 0.96OLEDGPIOs :
13. //OLED_SDA -- GPIO4
14. //OLED_SCL -- GPIO15
15. //OLED_RST -- GPIO16
16.
17.
18. #define MESH_PREFIX "HelloMyMesh"
19. #define MESH_PASSWORD "hellomymeshnetwork"
20. #define MESH_PORT 5555
21.
22. SSD1306 display(0x3c, 4, 15);
23. #define SS 18
24. #define RST 14
25. #define DIO 26
26. #define BAND 433.175E6 //915E6
27.
28. painlessMesh mesh;
29. // Send my ID every 10 seconds to inform others
30.
31. Task logServerTask(10000, TASK_FOREVER, []() {
32. DynamicJsonBuffer jsonBuffer;
33. JsonObject& msg = jsonBuffer.createObject();
34. msg["topic"] = "logServer";
35. msg["nodeId"] = mesh.getNodeId();
36. String str;
37. msg.printTo(str);
38. mesh.sendBroadcast(str);
39. // log to serial
40. msg.printTo(Serial);
41. Serial.printf("\n");
42. });
43.

```

```

44.
45. void setup() {
46.
47.
48. Serial.begin(115200);
49.
50. pinMode(25, OUTPUT); //Send success, LED will bright 1 second
51.
52. while (!Serial);
53.
54. pinMode(16, OUTPUT);
55. digitalWrite(16, LOW); // set GPIO16 low to reset OLED
56. delay(50);
57. digitalWrite(16, HIGH); // while OLED is running, must set GPIO16 in high
58.
59.
60. Serial.println("LoRa PainlessMesh Server");
61.
62. SPI.begin(5, 19, 27, 18);
63. LoRa.setPins(SS, RST, DIO);
64. if (!LoRa.begin(BAND)) {
65. Serial.println("Starting LoRa failed!");
66. while (1);
67. }
68.
69. Serial.println("LoRa Initial OK!");
70. // Initialising the UI will init the display too.
71. display.init();
72. display.flipScreenVertically();
73. display.setFont(ArialMT_Plain_10);
74.
75. display.clear();
76. display.setTextAlignment(TEXT_ALIGN_LEFT);
77. display.drawString(10, 5, "Mesh Server Node:");
78. display.display();
79.
80.
81.
82. mesh.setDebugMsgTypes(ERROR | CONNECTION | S_TIME);
83. mesh.init(MESH_PREFIX, MESH_PASSWORD, MESH_PORT, STA_AP, WIFI_AUTH_WPA2_PSK, 6
);
84. mesh.onReceive(&receivedCallback);
85. mesh.onNewConnection([](size_t nodeId) {
86. Serial.printf("New Connection %u\n", nodeId);
87. });
88. mesh.onDroppedConnection([](size_t nodeId) {
89. Serial.printf("Dropped Connection %u\n", nodeId);
90. });
91. // Add the task to the mesh scheduler
92. mesh.scheduler.addTask(logServerTask);
93. logServerTask.enable();
94. }
95.
96.
97. void loop() {
98. mesh.update();
99. }
100.
101.
102. void receivedCallback(uint32_t from, String &msg) {
103.
104. String tmp_string = msg.c_str();
105.
106. Serial.printf("logServer: Received from %u msg=%s\n", from, tmp_string);
107.
108. Serial.println("");
109. Serial.println("Sending LoRa packet: "+tmp_string);
110.

```

```

111. //เมื่อได้รับข้อมูลความจากใน mesh network ก็ส่งต่อผ่านไปยัง LoRa
112. LoRa.beginPacket();
113. LoRa.print(tmp_string);
114. LoRa.endPacket();
115.
116.
117. display.clear();
118. display.setTextAlignment(TEXT_ALIGN_LEFT);
119. display.drawString(10, 5, "Sending: "+tmp_string.substring(13,19));
120. display.drawString(10, 20, "Temp: "+tmp_string.substring(49,55));
121. display.drawString(10, 35, "Humid: "+tmp_string.substring(69,75));
122. // write the buffer to the display
123. display.display();
124.
125.
126. }

```

## Heltec ESP32 SX1276 (Gateway Node + LoRa)

Code:

```

1. #include <SPI.h>
2. #include <LoRa.h>
3. #include "SSD1306.h"
4.
5. #include <WiFi.h>
6. #include <WiFiClient.h>
7. #include <PubSubClient.h>
8.
9.
10.
11. const char* ssid = "xxxWiFi-SSID";
12. const char* password = "xxxWiFi Password";
13. const char* mqtt_server = "xxx.xxx.xxx.xxx"; //--- IP ว่า Domain ว่า Server MQTT
14.
15. long lastMsg = 0;
16. char msg[100];
17. int value = 0;
18.
19. WiFiClient espClient;
20.
21. void callback(char* topic, byte* payload, unsigned int length) {
22. Serial.print("Message arrived [");
23. Serial.print(topic);
24. Serial.print("] ");
25. for (int i = 0; i < length; i++) {
26. Serial.print((char)payload[i]);
27. }
28. Serial.println();
29. }
30.
31.
32.
33.
34. PubSubClient client(mqtt_server, 1883, callback, espClient);
35.
36.
37.
38.
39. // GPIO5 -- SX1278's SCK
40. // GPIO19 -- SX1278's MISO
41. // GPIO27 -- SX1278's MOSI
42. // GPIO18 -- SX1278's CS
43. // GPIO14 -- SX1278's RESET
44. // GPIO26 -- SX1278's IRQ(Interrupt Request)
45. //OLED pins to ESP32 0.96OLEDGPIOs
46. //OLED_SDA -- GPIO4

```

```

47. //OLED_SCL -- GPIO15
48. //OLED_RST -- GPIO16
49. SSD1306 display(0x3c, 4, 15);
50. #define SS 18
51. #define RST 14
52. #define DIO 26
53. #define BAND 433.175E6 //915E6
54.
55. void setup() {
56. Serial.begin(115200);
57.
58. while (!Serial);
59. pinMode(16, OUTPUT);
60. digitalWrite(16, LOW); // set GPIO16 low to reset OLED
61. delay(50);
62. digitalWrite(16, HIGH); // while OLED is running, must set GPIO16 in high
63. Serial.println("LoRa Receiver");
64.
65. SPI.begin(5, 19, 27, 18);
66. LoRa.setPins(SS, RST, DIO);
67. if (!LoRa.begin(BAND)) {
68. Serial.println("Starting LoRa failed!");
69. while (1);
70. }
71. Serial.println("LoRa Initial OK!");
72. // Initialising the UI will init the display too.
73.
74. display.init();
75. display.flipScreenVertically();
76. display.setFont(ArialMT_Plain_10);
77.
78. setup_wifi();
79. client.connect("ESP32Gateway", "joe1", "joe1");
80. client.setCallback(callback);
81. client.subscribe("command");
82.
83.
84.
85. }
86.
87. void setup_wifi() {
88. delay(10);
89. // We start by connecting to a WiFi network
90. Serial.println();
91. Serial.print("Connecting to ");
92. Serial.println(ssid);
93. WiFi.begin(ssid, password);
94. while (WiFi.status() != WL_CONNECTED) {
95. delay(500);
96. Serial.print(".");
97. }
98. Serial.println("");
99. Serial.println("WiFi connected");
100. Serial.println("IP address: ");
101. Serial.println(WiFi.localIP());
102. }
103.
104. void reconnect() {
105. // Loop until we're reconnected
106. while (!client.connected()) {
107. Serial.print("Attempting MQTT connection...");
108. // Attempt to connect
109. if (client.connect("ESP32Gateway")) {
110. Serial.println("connected");
111. // Once connected, publish an announcement...
112. client.publish("outTopic", "hello world");
113. // ... and resubscribe
114. client.subscribe("command");
115. } else {
116. Serial.print("failed, rc=");

```

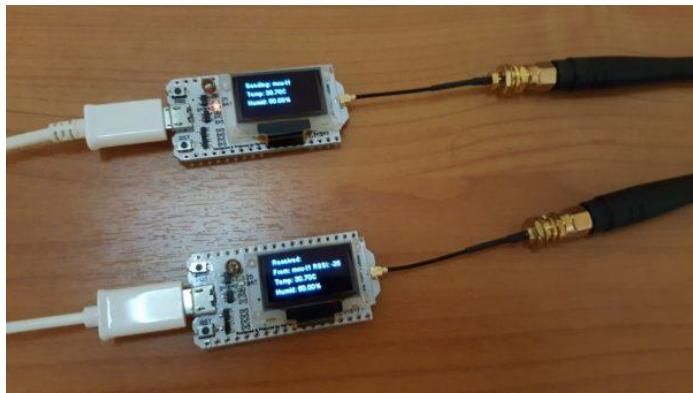
```

117. Serial.print(client.state());
118. Serial.println(" try again in 5 seconds");
119. // Wait 5 seconds before retrying
120. delay(5000);
121. }
122. }
123.
124.
125. void loop() {
126. String tmp_string, tmp_rssi;
127.
128. if (!client.connected()) {
129. reconnect();
130. }
131. client.loop();
132.
133. // try to parse packet
134. int packetSize = LoRa.parsePacket();
135.
136. if (packetSize) {
137. // received a packet
138. Serial.print("Received packet '");
139. // read packet
140. while (LoRa.available()) {
141. //Serial.print((char)LoRa.read());
142. tmp_string += (char)LoRa.read();
143. }
144. Serial.print(tmp_string);
145.
146. tmp_rssi = LoRa.packetRssi();
147.
148. // print RSSI of packet
149.
150.
151. Serial.println("' with RSSI " + tmp_rssi);
152.
153.
154. display.clear();
155. display.setTextAlignment(TEXT_ALIGN_LEFT);
156. display.drawString(10, 0, "Received:");
157. display.drawString(10, 15, "From: " + tmp_string.substring(13, 19) + " RSSI:
" + tmp_rssi);
158. display.drawString(10, 30, "Temp: " + tmp_string.substring(49, 55));
159. display.drawString(10, 45, "Humid: " + tmp_string.substring(69, 75));
160. // write the buffer to the display
161. display.display();
162.
163. tmp_string.toCharArray(msg, 100);
164. Serial.print("Publish message: ");
165. Serial.println(msg);
166. client.publish("env", msg); //ส่งข้อมูล Temp + Humidity ออกไปที่ Topic "env"
167.
168.
169. }
170. tmp_string = "";
171. tmp_rssi = "";
172.
173.
174. }

```

### ผลลัพธ์ที่ได้

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



ในอีกฟากหนึ่ง Gateway Node: Heltec ESP32 LoRa SX1278 เมื่อได้รับข้อมูลจาก LoRa ก็จะเอาข้อมูลนั้น Publish ต่อไปยัง MQTT Server ที่เราได้สร้างกันไว้ตั้งแต่ตอนที่ 3.5 ถ้าดูจากคลิปวิดีโอดูเหมือนว่าระยะเวลาที่ใช้แนบค่อนข้างเร็วมาก ซึ่งหลังจากนี้การจะนำข้อมูลจากภายใน Mesh Network ไปใช้งานก็ง่ายแล้ว จะเขียนลง Time Series Database เพื่อเอาไปแสดงผล หรือวิเคราะห์ข้อมูลต่อ ก็ไม่ยากแล้ว

### สรุป

เนื้อหาในตอนนี้จะเน้นในเรื่องของ Data Communication อีกรูปแบบหนึ่งนั่นก็คือ LoRa ในการนำมาใช้งานร่วมกับ Mesh Network ซึ่งอย่างที่เกริ่นไปตอนแรกจะครับ ถ้าผ่านตอนที่ 3 มาแล้วก็จะเขียนอยู่กับเราว่าจะไป bridge กับอะไรเพื่อส่งต่อข้อมูลภายใน Mesh Network ไปยังอีก Network หนึ่ง ถ้าตัว Server Node ของเราต้องเข้ากับ Ethernet Module ENC28J60 ก็สามารถเป็น Bridge ในตัวมันเองได้โดยโดยส่งข้อมูลผ่านสาย Lan ก็ได้ ดังนั้นที่เขียนมาหลายตอนมากก็เพื่ออย่างให้เข้าใจหลักการณ์ก่อน ที่เหลือก็จะเขียนอยู่กับการประยุกต์ใช้ให้เหมาะสมนั่นเองครับ ใช่ว่าจำเป็นจะต้องใช้งาน Mesh Network สำหรับงาน IoT ทุกเคส หรือว่าจำเป็นต้องใช้งาน LoRa เพื่อให้ได้ระยะใกล้ๆ บางครั้งต่อพ่วงอุปกรณ์ของเราเข้ากับ GPRS/LTE Module ก็อาจจะทำให้ได้ใกล้กว่าถึงชัยขอบของผู้ให้บริการมือถือได้เลยด้วยซ้ำ ใกล้กว่า NB-IoT ตอนนี้ด้วย

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

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

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

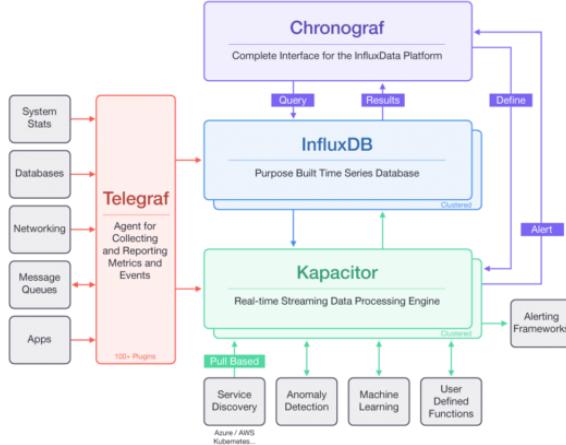
จริงๆเนื้อหาหลักของ Painlessmesh นั้นจบไปตั้งแต่ตอนที่ 3 แล้ว ที่เหลือจะเป็นลักษณะของการประยุกต์ใช้มากกว่า ไม่ว่าจะเป็นการ Bridge ด้วย LoRa หรือการติดตั้ง MQTT Server บน Google Cloud ซึ่งถ้าันบจากตอนล่าสุดตอนที่ 4 ถึงตอนนี้ก็ผ่านไป 4 เดือนละ ก็เลยจะมาจบ Series ของ Painlessmesh นี้ตามที่เคยเกริ่นไว้ว่าจะมีทั้งหมด 5 ตอนด้วยกัน โดยตอนนี้ก็จะเป็นเหมือนตอนแรกปลายทางละ เพราะจะเป็นการนำข้อมูลต่างๆนั้นไปเก็บอยู่บน Time Series Database ที่ชื่อว่า InfluxDB และนำไปแสดงผลบน Dashboard ของ Grafana

ก่อนเข้าไปสู่การติดตั้ง InfluxDB ซึ่งเป็น Time Series Database (TSDB) โดยที่เป็นการเก็บข้อมูลเหมือนเก็บ log ที่เก็บไปเรื่อยๆเรียงตามวันที่และเวลาต่อๆกันไป เช่นข้อมูลราคาหุ้น ข้อมูลอุณหภูมิ ความชื้นในแต่ละช่วงเวลา ซึ่ง InfluxDB ก็เป็นหนึ่งใน TSDB ที่ถูก Optimize มาให้ทำงานในลักษณะนี้ โดยเน้นการทำงานที่ง่าย เร็ว ตัดพวง business logic ต่างๆไปอยู่ที่ module ข้างนอกแทน ฉะนั้นในการโอนข้อมูลเข้ามาเพื่อเขียนลงฐานข้อมูลก็จะเร็วมากและใช้พื้นที่น้อยกว่าอีกด้วย

InfluxDB ก็มีทั้งส่วนที่เป็น Commercial และ Opensource ซึ่งข้อแตกต่างหลักๆของ version ที่เสียเงินกับ Opensource นั้นก็จะเป็นในเรื่องของการ support และก็เรื่องของ High Availability หรือการ Scale out ระบบเพื่อรับการทำ Clustering ระหว่างหลายๆ node ของ InfluxDB

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

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



โดยที่การทำงานก็จะเป็นอย่างรูปที่แสดงอยู่ด้านบนเลยครับ

- Telegraf จะเป็นตัวกลางในการรับข้อมูลจาก Source ต่างๆเข้ามาไม่ว่าจะเป็น Database อื่นๆหรือแม้กระทั่ง MQTT Message และรวมถึงดึงข้อมูลจาก InfluxDB และส่งกลับไปยังระบบอื่นด้วย ทำหน้าที่เหมือน ETL
- InfluxDB ก็เป็นหัวใจของ TICK Stack นี้เลยซึ่งก็คือ Time Series Database ของเรานั่นเอง เน้นเก็บข้อมูลอย่างเดียว
- Chronograf อันนี้จะเป็น GUI ที่ใช้ในการ Manage InfluxDB รวมถึงการทำ Dashboard ด้วย (แต่เดียวเราจะไปใช้ Grafana ทำ อิอิ)
- Kapacitor จะเป็นส่วนที่ทำในเรื่องของ Data Processing ต่างๆ เช่น เมื่อมีข้อมูลการรอสายโทรศัพท์ของลูกค้าที่โทรเข้ามาที่ Call Center เกินเท่านี้ให้ทำการ Alert แจ้งเตือนไปยังหัวหน้างาน ซึ่งก็สามารถนำ Module การวิเคราะห์ต่างๆเข้ามาใช้ได้ ไม่ว่าจะเป็นจากการตั้งค่าเอง หรือ Machine Learning ก็ได้

### InfluxDB Setup

โดยการติดตั้งนั้นจะเริ่มจาก InfluxDB กันก่อน โดยการเปิดหน้า SSH Command Windows ของ Google Cloud และพิมพ์คำสั่งต่อตามด้านล่างนี้เลยครับโดยสเตปก็จะเป็น Download & Install -> Start Service -> Verify (ใครที่อ่านตรงนี้แล้วง อาจต้องย้อนกลับไปอ่านที่ ตอนแรก 3.5-1 การติดตั้ง MQTT Server)

#### Download & Install

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

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

Start InfluxDB

```
sudo systemctl start influxdb
```

Verify

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

ถ้า InfluxDB ที่เราเพิ่งติดตั้งไปทำงานได้อย่างถูกต้องแล้วหลักๆ จะแสดงผลลัพธ์ตามบรรทัดล่างนี้เลยครับ

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

### Kapacitor Setup

มาถึง Module ที่สองที่เราจะทำการติดตั้งกัน เพื่อใช้ในการทำ Data Processing ซึ่งถ้าไม่ได้ใช้งานก็สามารถข้ามไปที่ Module ถัดไปได้ครับ

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
```

ซึ่งถ้าติดตั้งและ Service Start เรียบร้อยแล้ว ผลลัพธ์ที่ได้จะเป็นดังนี้ครับ

| ID | Type | Status | Executing Databases and Retention Policies |
|----|------|--------|--------------------------------------------|
|----|------|--------|--------------------------------------------|

### Telegraf Setup

Module Telegraf นี้เรียกได้ว่าเป็นอีกหัวใจหลักเลยก็ว่าได้ครับ เพราะจะเป็นตัวที่เก็บรวบรวม ดึงข้อมูล จาก Source ต่างๆเพื่อที่จะโอนเข้า InfluxDB ในตัวอย่างนี้เราจะเรียกเก็บ System Stat ของ VM Instance ของเรา กันด้วยเพื่อดู load ของ CPU / RAM กัน

Download & Install

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

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

Start Service

```
sudo systemctl start telegraf
```

## Verify

ขั้นตอนนี้จะแตกต่างจาก Module อื่นๆ อย่างที่เกริ่นกันไว้ เราจะดึงข้อมูล System Stat ของระบบของเรามา โดยการทำงานของ Telegraf จะทำงานผ่าน Plug-in ต่างๆ โดยขึ้นแรกให้เปิดไฟล์ /etc/telegraf/telegraf.conf จากนั้นให้ดูในส่วนของ Output Plugins (จะใช้โปรแกรม vi, pico , nano อันนี้ก็แล้วแต่สะดวกกันเลยครับ)

```

1. [[outputs.influxdb]]
2. ## The full HTTP or UDP endpoint URL for your InfluxDB instance.
3. ## Multiple urls can be specified as part of the same cluster,
4. ## this means that only ONE of the urls will be written to each interval.
5. # urls = ["udp://localhost:8089"] # UDP endpoint example
6. urls = ["http://localhost:8086"] # required
7. ## The target database for metrics (telegraf will create it if not exists).
8. database = "telegraf" # required
9.
10. ## Retention policy to write to. Empty string writes to the default rp.
11. retention_policy = ""
12. ## Write consistency (clusters only), can be: "any", "one", "quorum", "all"
13. write_consistency = "any"
14.
15. ## Write timeout (for the InfluxDB client), formatted as a string.
16. ## If not provided, will default to 5s. 0s means no timeout (not recommended).
17. timeout = "5s"
18. # username = "telegraf"
19. # password = "metricsmetricsmetrics"
20. ## Set the user agent for HTTP POSTs (can be useful for log differentiation)
21. # user_agent = "telegraf"
22. ## Set UDP payload size, defaults to InfluxDB UDP Client default (512 bytes)
23. # udp payload = 512

```

และในส่วนของ Input Plugins นั้นก็ควรจะมีหน้าตาลักษณะแบบนี้ ในการดึงค่า System Stat ออกมานำเพื่อไปเป็น Output ให้กับ InfluxDB

```

1. # Read metrics about cpu usage
2. [[inputs.cpu]]
3. ## Whether to report per-cpu stats or not
4. percpu = true
5. ## Whether to report total system cpu stats or not
6. totalcpu = true
7. ## If true, collect raw CPU time metrics.
8. collect_cpu_time = false
9.
10. # Read metrics about disk usage by mount point
11. [[inputs.disk]]
12. ## By default, telegraf gather stats for all mountpoints.
13. ## Setting mountpoints will restrict the stats to the specified mountpoints.
14. # mount_points = ["/"]
15.
16. ## Ignore some mountpoints by filesystem type. For example (dev)tmpfs (usually
17. ## present on /run, /var/run, /dev/shm or /dev).
18. ignore_fs = ["tmpfs", "devtmpfs"]
19.
20.
21. # Read metrics about disk IO by device
22. [[inputs.diskio]]
23.

```

```

24. ## By default, telegraf will gather stats for all devices including
25. ## disk partitions.
26. ## Setting devices will restrict the stats to the specified devices.
27. # devices = ["sda", "sdb"]
28. ## Uncomment the following line if you need disk serial numbers.
29. # skip_serial_number = false
30.
31.
32. # Get kernel statistics from /proc/stat
33. [[inputs.kernel]]
34. # no configuration
35.
36.
37. # Read metrics about memory usage
38. [[inputs.mem]]
39. # no configuration
40.
41.
42. # Get the number of processes and group them by status
43. [[inputs.processes]]
44. # no configuration
45.
46.
47. # Read metrics about swap memory usage
48. [[inputs.swap]]
49. # no configuration
50.
51.
52. # Read metrics about system load & uptime
53. [[inputs.system]]
54. # no configuration

```

ซึ่งจริงๆแล้วทั้งในส่วนของ Output/Input Plugins นั้นก็เป็นค่า Default ที่ติดตั้งมาพร้อมกับ Telegraf อยู่แล้วครับ เสร็จแล้วก็ทดลองรันคำสั่งนี้กันดู

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

ถ้า Telegraf ทำงาน ก็จะแสดงผลโดยวาร์ป์ด้วยภาษาเป็นรูปแบบ JSON ให้ได้เห็นกันเต็มหน้าจอเล

## Chronograf Setup

มาถึง Module สุดท้ายของ TICK Stack กันเลยครับสำหรับหน้า GUI เพื่อใช้ในการ Config และแสดงผล สำหรับผู้คนแล้วเอาไปทำบน Grafana น่าจะสะดวกกว่า แต่ไหนๆก็ใหญ่แล้ว Install กันให้ครบเลยดีกว่าครับ เป็นทางเลือก

## Download & Install

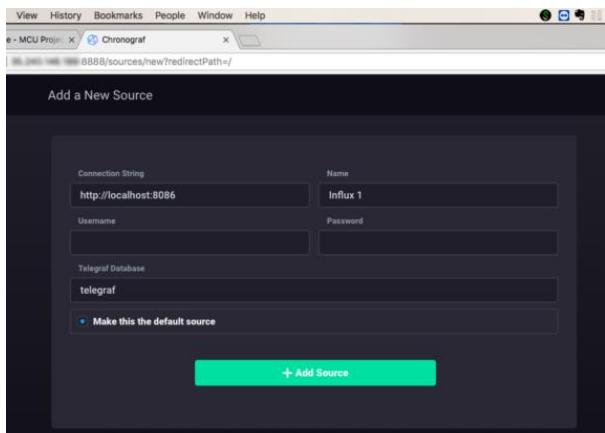
```
1. wget https://dl.influxdata.com/chronograf/releases/chronograf_1.4.0.0_amd64.deb
2.
3. sudo dpkg -i chronograf_1.4.0.0_amd64.deb
```

## Start Service

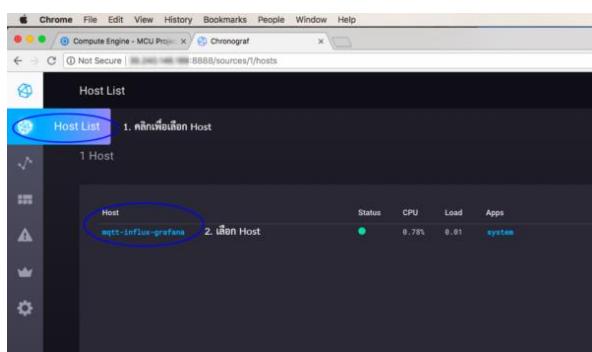
```
1. * အကွဲ အလောမေးမှု အနေဖူ ဓမ္မပေးခြင်း
```

## Verify

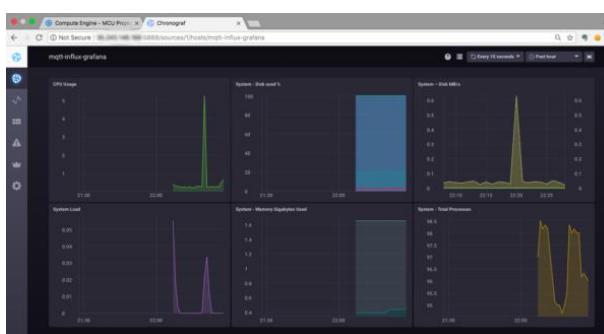
ขั้นตอนนี้ก็ให้เข้าไปที่ <http://localhost:8888>โดยเปลี่ยน local host เป็น ip หรือ hostname ของเรา ครับ ก็จะได้หน้า Web GUI ของ Chronograf ดังแสดงในรูปด้านล่างนี้ สำหรับใครที่ใช้ Google Cloud ก็อย่าลืม เข้าไปเช็คในส่วนของ Firewall Rule ให้ Allow Port 8888 ด้วยนะครับ วิธีการก็หาดูได้จากตอนที่ 3.5 ได้เลย



หลังจากนั้นก็คลิกที่ปุ่ม +Add Source เลยครับ ค่าที่อยู่ให้คง Default ไว้ เนื่องจากการติดตั้งครั้งแรก นั้น Default Security ของ InfluxDB นั้นจะไม่มี Username/Password ซึ่งเดียวเราจะได้มาม config กันในช่วงต่อไปครับ ซึ่งหลังจาก Add Source แล้วก็จะเข้าสู่หน้าจอถัดไป ให้ทำการเลือก Host ที่เราใช้ Plugin ของ Telegraf ในการดึง System Stat มาใส่ไว้ใน InfluxDB



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



แต่ยังไม่จบครับ เพราะ TICK Stack นั้นมีอยู่ด้วยกัน 4 Module นี้เราเพิ่งจะใช้งานไปแค่ 3 Module เอง ยังเหลือในส่วนของ Kapacitor ที่ใช้ในการทำ Data Processing/Analysis และ Alert ต่างๆ จากหน้า Dashboard System Stat ของเราคลิกที่ Config เพื่อ connect ไปยัง Kapacitor แต่ผมจะขอหยุดไว้เท่านี้ก่อน ดีกว่า เพราะถ้าเราทำใน Grafana นั้นในส่วนของ Dashboard และ Alert เราสามารถทำได้จาก Grafana ในที่

เดียวกันเลย เราจึงมักเห็นการจับคู่กันเฉพาะในส่วนของ MQTT -> Telgraf-> InfluxDB->Grafana เรียกได้ว่า เป็นสูตรสำเร็จสำหรับการเก็บข้อมูลมาแสดงผลของงาน IOT เลยครับ

แต่ก่อนอื่น เรา มาเซทระบบของเราให้เข้าที่เข้าทางกันก่อนดีกว่า เพราะก่อนหน้านี้ที่เรา Install InfluxDB ไปค่า Default ของระบบยังไม่มีในส่วนของ User/Password และการ Authen ต่างๆ ขึ้นตอนแรกเราจะสร้าง User Admin ของระบบกันด้วยคำสั่ง influx และตามด้วยคำสั่งด้านล่างนี้ครับ

```

1. CREATE USER "admin" WITH PASSWORD 'admin_passwd' WITH ALL PRIVILEGES
2.
3.
4. ในส่วนของ admin_passwd ก็เปลี่ยนເຄາມໃຊ້ອນໄດ້ເລຍຮັບ
```

หลังจากนั้นก็ใช้คำสั่ง show users ในการเรียกดู user ทั้งหมดที่มีอยู่ ตามรูปด้านล่างนี้

```
mqtt-influx-grafana: /etc/influxdb
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 'admin_passwd' 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-...@instances/mqtt-influx-g...
 mqtt-influx-grafana:/etc/influxdb$ influx -username
 influx-grafana:/etc/influxdb$ influx -password
 influx-grafana:/etc/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

```
Secure | https://ssh.cloud.google.com/projects/mcu-office-home-...@instances/mqtt-influx-g...
 mqtt-influx-grafana:/etc/influxdb$ influx -username admin -password
 influx-grafana:/etc/influxdb$ influx -username admin -password
 influx-grafana:/etc/influxdb$ influx
 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

```
1. [[inputs.mqtt_consumer]]
2. servers = ["xxx.xxx.xxx.xxx:1883"] #<-- IP ของ mqtt server
3. qos = 0
4. connection_timeout = "30s"
5. topics = ["env",] #<-- topic ที่เราต้องการดึงข้อมูลมา
6. username = "xxx" #<-- username ของ mqtt server
7. password = "xxx" #<-- password ของ mqtt server
8. data_format = "influx" #<-- รูปแบบของ Data ที่ต้องการดึง
```

มาตรฐานส่วนของ influx line format กันบ้าง ซึ่งรูปแบบของข้อมูลที่ Telegraf รองรับนั้นก็มีหลายรูปถึง JSON ด้วย ถ้าสนใจจะใช้ format อื่นๆ คุณก็สามารถค้นได้เลยครับ telegraf input data formats

เนื่องจากตอนนี้เราจะมีการเขียนข้อมูลลงฐานข้อมูลของ InfluxDB 2 ตัวด้วยกัน โดยตัวแรกก่อนหน้านี้คือ Default ที่มาจากการติดตั้ง Telegraf ซึ่งเป็นข้อมูลพวก System Stat อีกอันก็จะเป็นข้อมูลที่เราได้มาจาก Node ที่ส่งข้อมูลมาที่ MQTT Server (ในที่นี้จำลองส่งข้อมูลอุณหภูมิเข้ามา) จะนั้นเราต้องบอก Telegraf ว่าข้อมูลไหนจะเก็บลงฐานข้อมูลไหน โดยการใช้ namedrop, namepass ตามตัวอย่างด้านล่างนี้ครับ

```

1. [[outputs.influxdb]] #<-- ໃໝ່ເຂົ້າມີຂໍ້ມູນ System Stat ລົງ InfluxDB ໂດຍເກັບທຳນີ້ຂໍ້ມູນ
2. telegraf
3. urls = ["http://localhost:8086"] # required
4. database = "telegraf" # required
5. retention_policy = ""
6. write_consistency = "any"
7. timeout = "5s"
8. username = "xxx" #<-- username ຂອງ InfluxDB
9. password = "xxx" #<-- password ຂອງ InfluxDB
10. namedrop = ["env*"] #<-- ໃຫ້ drop ຂໍ້ມູນທີ່ຂຶ້ນດັ່ງຕະຫຼາມ env ຊື່ນີ້ເປັນ measurement
11.
12. [[outputs.influxdb]] #<-- ໃໝ່ເຂົ້າມີຂໍ້ມູນ System Stat ລົງ InfluxDB ໂດຍເກັບທຳນີ້ຂໍ້ມູນ
13. telegraf
14. urls = ["http://localhost:8086"]
15. database = "envdb" # required
16. retention_policy = ""
17. write_consistency = "any"
18. timeout = "5s"
19. username = "xxx" #<-- username ຂອງ InfluxDB
20. password = "xxx" #<-- password ຂອງ InfluxDB
 namepass = ["env*"] #<-- ຂໍ້ມູນທີ່ຂຶ້ນດັ່ງຕະຫຼາມ env ຊື່ນີ້ເປັນ measurement ໃຫ້ເຂົ້າລົງຮຽນຂໍ້ມູນ

```

ຈາກນີ້ທ່ານກ່ຽວຂ້ອງ Restart Service

```

1. $ systemctl daemon-reload
2. $ sudo systemctl restart telegraf
3. $ sudo systemctl status telegraf

```

ສ່ວນ influx line format ນັ້ນກໍຈ່າຍທີ່ສຸດເລຍຄົບ ຍກຕ້ວອຍ່າງ ຕາມດ້ານລ່າງເປັນຂໍ້ມູນທີ່ Nodemcu ສ່າງໄປທີ່ MQTT Server

env,location=RST temp\_in=25.60,temp\_out=25.40

ຫຸ້ວ່າໃຊ້ command line ຂອງ influx ເພີ້ມຂໍ້ມູນສ່າງເຂົ້າໄປທີ່ MQTT Server ກໍທ່ານກ່ຽວຂ້ອງ inset ຂໍ້ມູນເອງໂດຍໃໝ່ຄໍາສັ່ງນີ້ທີ່ທີ່

command line ຂອງ influx ເລຍຄົບ INSERT env,location=RST temp\_in=25.60,temp\_out=25.40

ສິ່ງກໍເກັບຂໍ້ມູນໃນ InfluxDB ຈະແຍກເປັນ Measurement, TAG Key – TAG Value ແລະ Field Key – Field Value ຈາກຕ້ວອຍ່າງຂໍ້ມູນຂ້າງບັນນິ້ນ ມີເຂົ້າຫ້ານ້າ command line ແລ້ວໃໝ່ຄໍາສັ່ງແສດງຄ່າດັ່ງຕ່ອງໄປນີ້

```

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 databases
name: databases
name: env
name: envdb
name: measurements
name: topics
> 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 ມີແຄ່ຕ້ວເດືອກວິກີ່ອ env

TAG Key ປະກອບໄປດ້ວຍ host, location, ແລະ topic

ແລະ temp\_in, temp\_out ເປັນ Field Key

## Grafana

ขั้นตอนการติดตั้ง Grafana ซึ่งโดยปกติสามารถติดตั้งได้ผ่าน apt-get install แต่บางเครื่องนั้นอาจจะไม่ได้มีข้อมูล repository ของ Grafana อยู่ด้วยฉะนั้นเราต้องทำการเพิ่มรายการของ Grafana เข้าไปในลิสต์ก่อนที่จะทำการติดตั้ง

```

1. curl https://packagecloud.io/gpg.key | sudo apt-key add -
2.
3.
4. จากนั้นตามด้วยคำสั่ง
5.
6.
7. sudo add-apt-repository "deb https://packagecloud.io/grafana/stable/debian/
stretch main"
8.
9.
10.
11. 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

```

mqtt-influx-grafana: /var/log/mosquitto
Secure | https://ssh.cloud.google.com/projects/mcuu-office-home
grafana-server.service - Grafana instance
 Loaded: loaded (/etc/lib/systemd/system/grafana-server.service; disabled; vendor preset: enabled)
 Active: active (running) since Sun 2018-08-26 06:13:03 UTC; 37s ago
 Tasks: 7 (limit: 1997)
 Process: 17303 (grafana-server)
 CGroup: /user.slice/grafana-server.service
 └─17303 /usr/sbin/grafana-server --config=/etc/grafana/grafana.ini --pidfile=/var/run/grafana/grafana.pid cfg:default.paths.log

Aug 26 06:13:05 mqtt-influx-grafana grafana-server[17303]: >2018-08-26T06:13:05+0000 lvl:info msg="Initializing InternalMetricsService" logger=server
Aug 26 06:13:05 mqtt-influx-grafana grafana-server[17303]: >2018-08-26T06:13:05+0000 lvl:info msg="Initializing AlertingService" logger=server
Aug 26 06:13:05 mqtt-influx-grafana grafana-server[17303]: >2018-08-26T06:13:05+0000 lvl:info msg="Initializing HTTPServer" logger=server
Aug 26 06:13:05 mqtt-influx-grafana grafana-server[17303]: >2018-08-26T06:13:05+0000 lvl:info msg="Initializing ConfigurationService" logger=server
Aug 26 06:13:05 mqtt-influx-grafana grafana-server[17303]: >2018-08-26T06:13:05+0000 lvl:info msg="Initializing MetricsService" logger=server
Aug 26 06:13:05 mqtt-influx-grafana grafana-server[17303]: >2018-08-26T06:13:05+0000 lvl:info msg="Initializing NotificationsService" logger=server
Aug 26 06:13:05 mqtt-influx-grafana grafana-server[17303]: >2018-08-26T06:13:05+0000 lvl:info msg="Initializing ProvisioningService" logger=server
Aug 26 06:13:05 mqtt-influx-grafana grafana-server[17303]: >2018-08-26T06:13:05+0000 lvl:info msg="Initializing RenderingService" logger=server
Aug 26 06:13:05 mqtt-influx-grafana grafana-server[17303]: >2018-08-26T06:13:05+0000 lvl:info msg="Initializing StreamService" logger=server
Aug 26 06:13:05 mqtt-influx-grafana grafana-server[17303]: >2018-08-26T06:13:05+0000 lvl:info msg="HTTP Server Listen" logger=http.server address
lines 1-17/19 [ctrl-C]

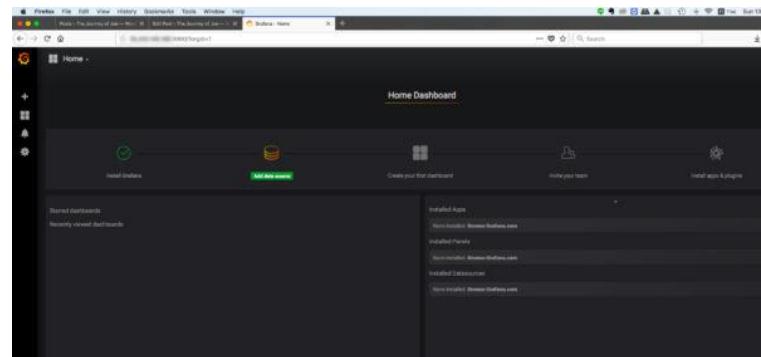
```

จากนั้นก็ตั้งค่าให้ Grafana เริ่มต้นทุกครั้งถ้ามีการรีบูตด้วยคำสั่ง `sudo systemctl enable grafana-server`

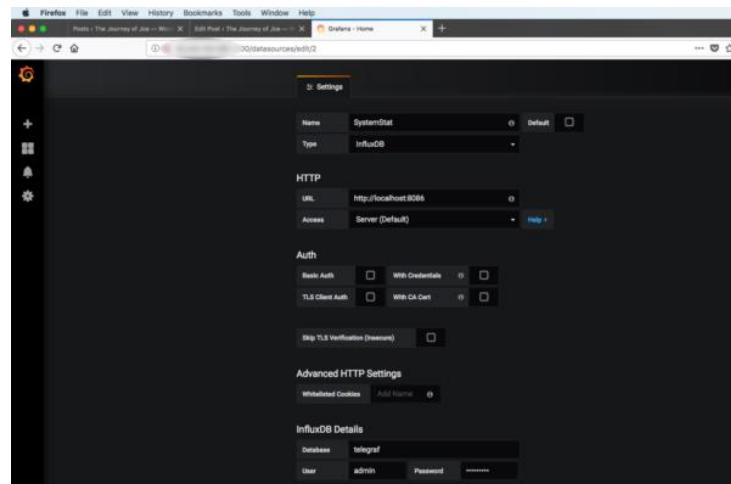
ซึ่งค่า Default Port ของ Grafana นั้นคือ port 3000 ดังนั้นถ้าใครใช้ Google Cloud ก็อย่าลืมไป allow port ให้ connection จากข้างนอกสามารถเข้าถึง VM Instance ของเราจาก port 3000 ได้ด้วยนะครับ ซึ่งการใช้งานก็เข้าได้จากหน้า Browser เลย



โดยที่ Default Username/Password ตั้งต้นเลยก็คือ admin/admin ครับ เมื่อ login เข้าไปแล้วระบบ ก็จะให้เราเปลี่ยน password ของ admin ใหม่ทันทีที่จะเข้าสู่หน้าถัดไป ซึ่ง Grafana นั้นก็ออกแบบขั้นตอนเป็นส เตปมาให้อย่างดีเมื่อติดตั้งเสร็จก็ เพิ่ม Datasource จากนั้นก็สร้าง Dashboard ครับ จะใช้คนเดียวก็ไม่ได้ ก็สามารถ Invite คนอื่นเข้ามาใช้งานร่วมกันได้ รวมถึงรองรับ Plugin ต่างๆด้วย



เราจะมา Add Data source และกัน ซึ่งก็คือ Data source System Stat ที่ได้มาจากการดึงข้อมูลของ Telegraf ในตอนต้นนั่นเอง ให้คลิก Add Data Source จากนั้นก็ใส่ข้อมูลตามที่เรา config ไว้



หลังจากที่เราเพิ่ม Data Source แรกของเราไปแล้ว ขั้นต่อไปก็เป็นการสร้าง Dashboard ของเรากัน โดยเข้าไปที่ New Dashboard และเพิ่ม Graph เข้าไปยัง Dashboard แรกของเรา



จากนั้นคลิกที่ชื่อ Panel แล้วเลือก Edit เพื่อทำการเลือก Data Source ว่าจะเอาข้อมูลไหนมาสร้างกราฟ



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



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

เป็นเนื้อหาการเขียนบล็อกที่เป็น Series ชุดยาวหลายตอนเลยที่เดียว มีอุปสรรค ออกไปเนื้อหาอื่นบ้านตามรายทาง เนื่องจากคนเขียนเอาแต่ใจ อ้อ เนื้อหาตอนถัดไปจะเป็นอะไรก็คงต้องติดตามกันต่อไปครับ มีสาระบ้าง ไม่มีสาระบ้าง แล้วเจอกันตอนต่อไปครับ

### Arm-Pelion Full Stack IoT Platform

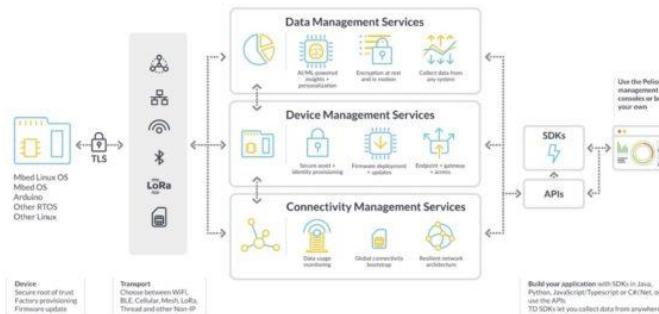
- <https://www.techtalkthai.com/arm-pelion-full-stack-iot-platform/>

### รู้จัก ARM Pelion แพลตฟอร์ม IoT จาก ARM จัดการทุกอย่างครบจบในที่เดียว

เมื่อวันที่ 30 กรกฎาคมที่ผ่านมา ทาง ARM ได้จัด IoT Workshop ขึ้นเพื่อนำเสนอและมุ่งตั้งๆของการนำเทคโนโลยี IoT ไปใช้ในธุรกิจ ทีมงาน TechTalkThai ได้เข้าไปร่วมงาน และทำความรู้จักกับ ARM Pelion IoT Platform จึงอยากจะมาเล่าให้ผู้อ่านฟังกันคร่าวๆว่าเจ้าแพลตฟอร์ม IoT นี้มีความสามารถอย่างไร และเหมาะสมกับธุรกิจแบบไหนบ้าง

IoT นั้นเป็นหนึ่งในเทคโนโลยีที่หลายองค์กรยกให้เป็นยุทธศาสตร์ในปี 2019 จากความสามารถในการรวบรวมข้อมูล ซึ่งนับว่าเป็นวัตถุดิบในการทำธุรกิจที่ขาดไม่ได้เลยในยุคปัจจุบัน จนถึงตอนนี้ หลายองค์กรอาจเริ่มต้นกับ IoT กันบ้างแล้ว แต่ความท้าทายใหม่ที่ธุรกิจมักจะเผชิญกับคือการจัดการและสกัดระบบ IoT ให้ใช้งานเก็บข้อมูลได้จริงเต็มประสิทธิภาพ ปลอดภัย และมีระบบจัดการที่ดี ดังนั้นจึงมีการพัฒนาแพลตฟอร์ม IoT ขึ้น เพื่อช่วยธุรกิจในการแก้ปัญหานี้

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 ส่วน จะทำงานร่วมกันภายใต้ระบบบรักษาความปลอดภัย ซึ่งเป็นหลักสำคัญที่สุดในการพัฒนาผลิตภัณฑ์ทุกๆตัวของ Arm



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

Pelion จะช่วยให้ธุรกิจจัดการกับเครือข่าย อุปกรณ์ในเครือข่าย และข้อมูลที่เก็บมาได้ง่ายขึ้น โดยสามารถทำงานร่วมกับอุปกรณ์ ระบบเครือข่าย คลาวด์ และข้อมูลได้หลากหลายรูปแบบ อีกทั้งยังมีความปลอดภัย และสามารถช่วยในการนำข้อมูลไปวิเคราะห์และแสดงผลเบื้องต้นได้ด้วย

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

### **Connectivity Management**

การเชื่อมต่อในเครือข่าย IoT นั้นมีอยู่หลายรูปแบบ และมีรายละเอียดปลีกย่อยที่ธุรกิจจะต้องจัดการอยู่พอกสมควร Pelion จะช่วยให้องค์กรสามารถจัดการการเชื่อมต่อได้อย่างมีประสิทธิภาพ ปลอดภัย และพร้อมต่อการสแกลเครือข่ายขึ้นไปถึงระดับโลก โดย Connectivity Management ของ Pelion มีความสามารถที่น่าสนใจดังนี้

### **Global Cellular**

Pelion จะช่วยให้อุปกรณ์ IoT สามารถเชื่อมต่อผ่านเครือข่ายได้ไม่ว่าอุปกรณ์นั้นจะอยู่ที่ใดในโลก ผ่าน เวนเดอร์เพียงเจ้าเดียว โดยกลไกของ Pelion จะช่วยเชื่อมต่อสัญญาณจากชิมของอุปกรณ์ไปยังเครือข่ายท้องถิ่นที่ Pelion ได้ทำข้อตกลงไว้ ลดภาระความปวดหัวในการติดต่อกับผู้ให้บริการเครือข่ายในแต่ละประเทศ

### **Protocol เชื่อมต่อทั้ง IP และ Non-IP**

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

### **ใช้ได้ทั้ง eSIM และซิมแบบปกติ**

ธุรกิจสามารถสั่งผลิตอุปกรณ์ที่มีระบบ eSIM ผ่าน ARM ได้ตามต้องการ โดย eSIM ที่ติดมากับอุปกรณ์ นั้นจะรองรับการเชื่อมต่อกับเครือข่ายกว่า 600 เครือข่ายทั่วโลก และหากต้องการเปลี่ยนเครือข่าย ก็สามารถตั้งค่าใหม่ได้ภายหลัง และในส่วนของซิมแบบปกติเอง Pelion ก็ให้บริการซิมการ์ดในทุกขนาด อีกทั้งยังมีแผนที่จะ พัฒนาไปจนถึง iSIM ที่มีขนาดเล็กกว่า eSIM หากด้วย

### **Network Infrastructure**

Pelion ได้พัฒนาโครงสร้างพื้นฐานของเครือข่ายให้สามารถทำงานร่วมกับผู้ให้บริการเครือข่ายทั่วโลกได้อย่างมีประสิทธิภาพสูงสุด โดยมีทั้งความเสถียร ยืดหยุ่น และเป็นไปตามกฎข้อบังคับด้านข้อมูลของแต่ละประเทศ ใน การใช้ Pelion ผู้ใช้จะสามารถเลือกได้ว่าจะส่งข้อมูลจากอุปกรณ์ไปยังแอปพลิเคชันผ่านเทคโนโลยีใด เช่น IPSEC, Open VPN, ผู้ให้บริการ Cloud, หรือทางเชื่อมที่ธุรกิจเช่ามาใช้โดยเฉพาะ (Leased Line)

## Device Management

Device Management ของ Pelion นั้นจะช่วยให้องค์กรสามารถจัดการกับอุปกรณ์และการเชื่อมต่อกับอุปกรณ์ผ่านซอฟต์แวร์ได้โดยสะดวก ไม่ว่าจะเป็นการเขียนซอฟต์แวร์แบบ Embedded หรือว่าการเขียนแอปพลิเคชันด้านบนอย่าง Web App ก็ตาม



Device Management ของ Pelion นี้รองรับการทำงานร่วมกับฮาร์ดแวร์ที่หลากหลาย ไม่ว่าจะเป็น อุปกรณ์แบบ Bare metal (มีระบบเชื่อมต่อที่เรียกว่า Edge รองรับ) และการทำงานร่วมกับระบบปฏิบัติการทั้ง Mbed OS และ Linux

## Data Management

เป้าประสงค์หลักของการจัดตั้งระบบ IoT นั้นคือการสร้างระบบจัดเก็บข้อมูลที่จะช่วยให้องค์กรสามารถเรียกข้อมูลเหล่านั้นขึ้นมาวิเคราะห์เป็นความรู้ที่มีประโยชน์ต่อธุรกิจได้ แน่นอนว่า Pelion ย่อมไม่ลืมความสำคัญของส่วนนี้ จึงได้พัฒนาระบบจัดการข้อมูลครบวงจรที่จะช่วยตั้งแต่การจัดเก็บ นำข้อมูลมาใช้ตัดสินใจแบบ Real-time และรักษาความปลอดภัยและความเป็นส่วนตัวของข้อมูล โดยมีกลไกรองรับการสเกลเต็มที่ ทำให้องค์กรไม่ต้องกังวลว่าระบบจะทำงานได้เยี่ยมหากมีข้อมูลหรืออุปกรณ์ในเครือข่าย IoT เพิ่มมากขึ้นเมื่อเวลาผ่านไป

โซลูชันหลักของส่วนนี้ คือ ARM Treasure Data ซึ่งเป็นซอฟต์แวร์จัดการและวิเคราะห์ข้อมูลที่เชื่อมต่อกันจาก Pelion ได้จบครบในตัวเดียว โดยมีเครื่องมือต่างๆพร้อมให้เลือกใช้งาน เช่น ระบบ Predictive Analytics การสร้าง Customer View 360 องศาจากข้อมูลการใช้งาน การวิเคราะห์ข้อมูลเพื่อ Cross-sell และ Upsell และการสร้างระบบ Recommendation เป็นต้น ซึ่งโซลูชันนี้หลายๆองค์กรก็ได้นำไปใช้งานเพิ่มประสิทธิภาพให้กับการทำงานในอุตสาหกรรมมากมาย เช่น อุตสาหกรรมค้าปลีก อุตสาหกรรมพลังงาน อุตสาหกรรมการผลิต และอุตสาหกรรมอื่นๆอีกมาก

แพลตฟอร์ม Pelion นั้นปัจจุบันได้มีการนำไปใช้งานกับระบบ IoT ทั้งในโปรเจกต์ขนาดใหญ่และขนาดเล็ก เช่น ระบบ IoT ในการดูแลสัตว์น้ำผ่านเซ็นเซอร์รับข้อมูลจากเสียง ระบบตรวจสอบสถานะการทำงานของ

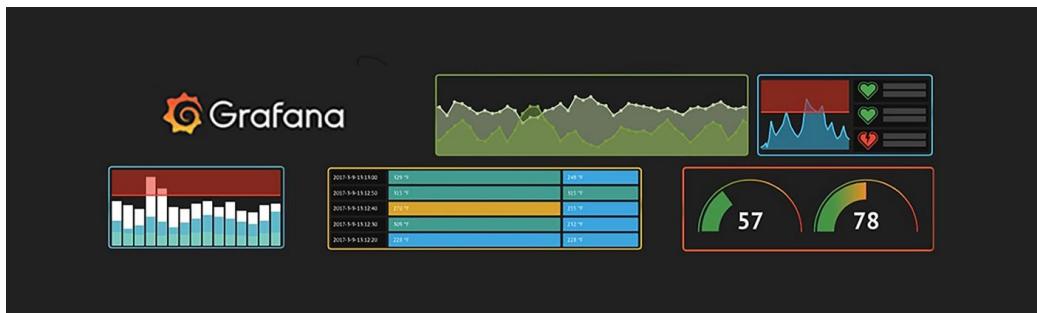
เครื่องจักรในโรงงาน ระบบจัดการคลังสินค้า และระบบซึ่งเป็นส่วนประกอบของ Smart City เช่น ที่จอดรถ อัจฉริยะ และเสาไฟฟ้าที่เปิดปิดตามความเคลื่อนไหวของคน และสามารถควบคุมได้จากระบบส่วนกลาง เป็นต้น

ท่านใดที่สนใจอยากศึกษาเกี่ยวกับ Pelion เพิ่มเติม สามารถเข้าไปอ่านเกี่ยวกับกรณีศึกษาการใช้งานใน อุตสาหกรรมได้ที่ <https://www.arm.com/products/iot/pelion-iot-platform> และหากต้องการข้อมูลเชิง เทคนิคโดยละเอียด สามารถอ่าน Document เต็มๆตามลิงก์นี้ <https://www.pelion.com/docs/> สำหรับในประเทศไทย ARM ได้จับมือเป็นพาร์ทนอร์กับ Advantech ใน การให้บริการด้านต่างๆ ท่านที่สนใจ สามารถติดต่อเพื่อพูดคุยถึงโซลูชันและผลิตภัณฑ์เกี่ยวกับ IoT ของ ARM ได้ที่อีเมล chanchai.p@advantech.com

## Grafana Dashboard

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

### ทำความรู้จักกับ Grafana Dashboard



Grafana คือ open source Dashboard tool เรียกง่าย ๆ ก็คือเครื่องมือในการสร้าง Dashboard ฟรี นั้นเอง

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

### จุดเด่นของ Grafana

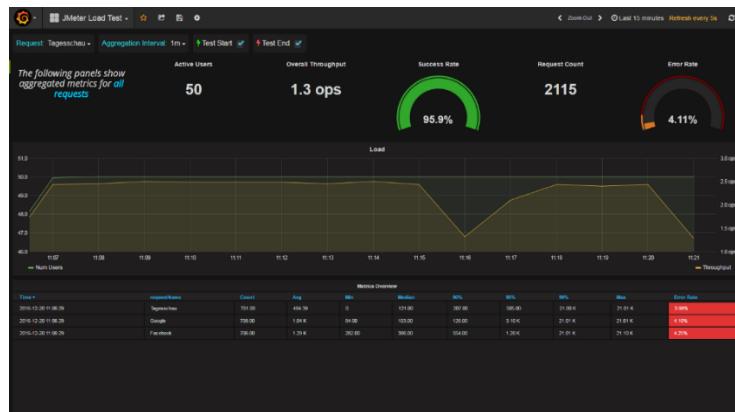
- เน้นการนำเสนอ Metrics ที่เฉพาะเจาะจง เช่น CPU, Memory หรือ I/O ในรูปแบบของกราฟ Time series
- มี Role-based access ในการจัดการ user ในการเข้าใช้งานให้ในตัว
- ความยืดหยุ่นในการใช้งาน มี option ให้เลือกใช้จำนวนมาก
- รองรับ datasource ที่หลากหลายและมี query editor ที่สำหรับ datasource นั้นๆ

## ตัวอย่างการใช้งาน Grafana Dashboard

- Monitoring Server ใช้งานร่วมกับ Influxdb และ Telegraf



- Monitoring Realtime result สำหรับ Jmeter ใน non-gui mode



## การติดตั้ง Grafana Dashboard

- ดาวน์โหลด ที่นี่ <https://grafana.com/grafana/download>

### สำหรับ Windows (x64)

- สามารถใช้ grafana-server.exe เพื่อเริ่มใช้งานได้ทันที
- กรณีต้องการระบุ custom config ดูรายละเอียด ที่นี่ <https://grafana.com/docs/grafana/v7.5/administration/configuration/>

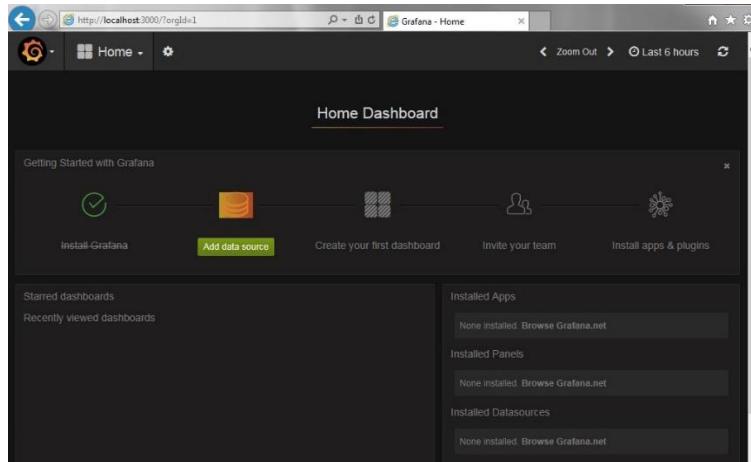
```
C:\Windows\system32\cmd.exe - bin\grafana-server.exe --config config\custom.ini
+ [32nINFO+{0m[07-28 12:33:40] Starting Grafana +[32nlogg
er+[0m=main +[32nversion+[0m=4.4.1 +[32ncommit+[0m=6a9f8caa4 +[32ncompiled+[0m=2
017-07-05T14:15:04+0700
+ [32nINFO+{0m[07-28 12:33:40] Config loaded from +[32nlogg
+ [32nINFO+{0m[07-28 12:33:40] Path Home +[32nlogg
+ [32nINFO+{0m[07-28 12:33:40] Path Data +[32nlogg
+ [32nINFO+{0m[07-28 12:33:40] Path Logs +[32nlogg
+ [32nINFO+{0m[07-28 12:33:40] Path Plugins +[32nlogg
+ [32nINFO+{0m[07-28 12:33:40] Path Plugins +[32nlogg
+ [32nINFO+{0m[07-28 12:33:40] Initializing DB +[32nlogg
+ [32nINFO+{0m[07-28 12:33:40] Starting DB migration +[32nlogg
+ [32nINFO+{0m[07-28 12:33:40] Starting DB migration +[32nlogg
+ [32nINFO+{0m[07-28 12:33:40] Executing migration +[32nlogg
+ [32nINFO+{0m[07-28 12:33:40] Executing migration +[32nlogg
+ [32nINFO+{0m[07-28 12:33:40] Skipping migration condition not fulfilled +[32nlog
gger+[0m=account +[32ninfo+[0m="copy data account to org"
+ [32nINFO+{0m[07-28 12:33:40] Executing migration +[32nlogg
+ [32nINFO+{0m[07-28 12:33:40] Starting plugin search +[32nlogg
+ [32nINFO+{0m[07-28 12:33:41] Initializing Alerting +[32nlogg
+ [32nINFO+{0m[07-28 12:33:41] Initializing CleanUpService +[32nlogg
+ [32nINFO+{0m[07-28 12:33:41] Initializing Stream Manager +[32nlogg
+ [32nINFO+{0m[07-28 12:33:41] Initializing HTTP Server +[32nlogg
+ [32nINFO+{0m[07-28 12:33:41] http://server +[32naddress+[0m=0.0.0.0:3000 +[32nprotocol+[0m=http +[32nsub
Url+[0m= +[32nsocket+[0m=

```

### สำหรับ Mac (Via Homebrew)

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

- เมื่อทำการติดตั้งและ start service เรียบร้อยแล้ว เริ่มต้นใช้งานโดย default port ของ grafana คือ 3000
- เข้าใช้งานโดย <http://localhost:3000> และ user/password เริ่มต้นคือ admin/admin



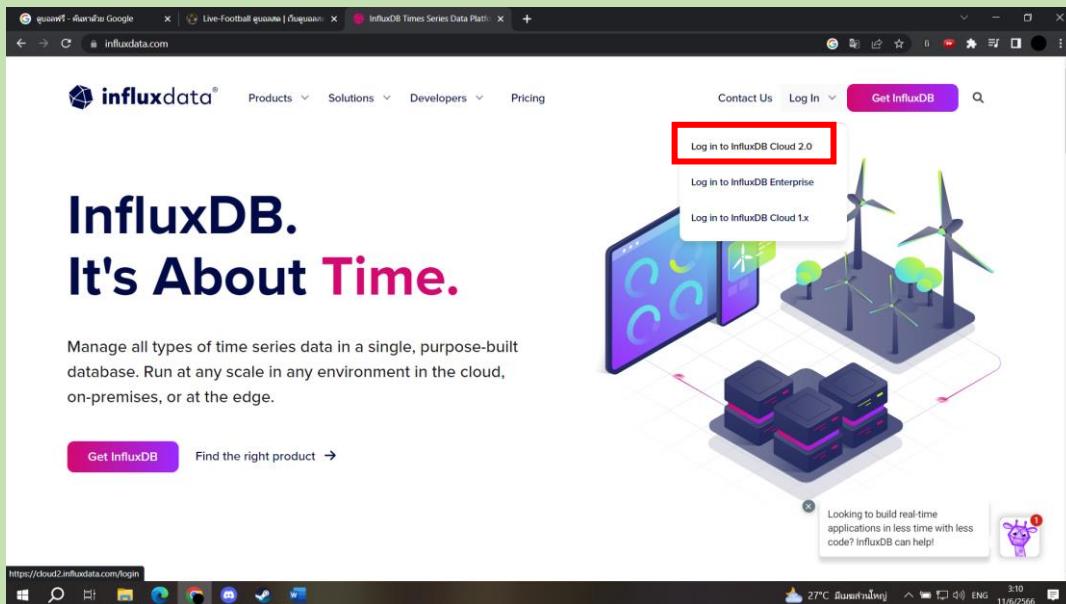
เท่านี้ก็สามารถเริ่มต้นใช้งาน Grafana Dashboard ได้แล้ว ครั้งหน้าจะมาแนะนำการใช้งานร่วมกับ Influxdb ใน การ Monitoring Server และ Monitoring realtime jmeter ~~~\*

#### 4/4. การทดสอบ

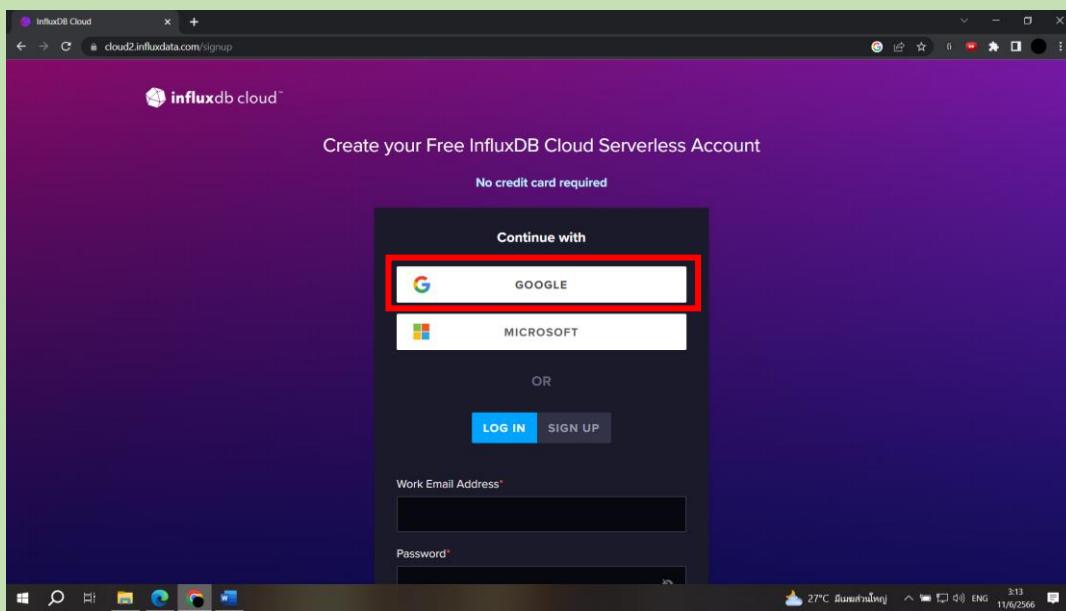
- ให้ทำการทดสอบและเขียนขั้นตอนการทดสอบ โดยใช้ ESP32 ส่งข้อมูลไปยัง MQTT Broker และใช้ Grafana .ในการอนิเเตอร์ข้อมูล โดยปรับแก้การทดสอบจาก <https://gabrieltanner.org/blog/grafana-sensor-visualization>

#### Visualize Sensor data using Grafana and InfluxDB

##### 1. Login to InfluxDB Cloud 2.0



##### 2. Login ผ่าน google



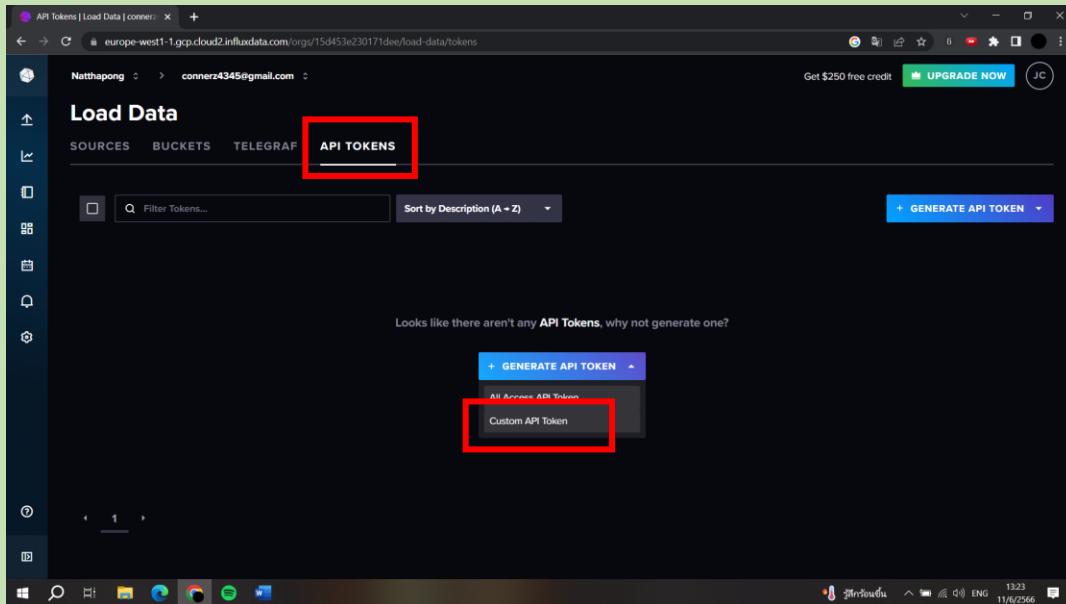
### 3. สร้าง Buckets

The screenshot shows the InfluxDB Cloud 'Get Started' interface. On the left, there's a sidebar with icons for Sources, Buckets (which is highlighted with a red box), Telegraf, and API Tokens. Below the sidebar, there are four programming language cards: Python, Node.js, Go, and Arduino. Under the 'Buckets' section, there are two cards: 'InfluxDB CLI' and 'Server Agent (Telegraf)'. A red box highlights the 'Buckets' icon in the sidebar.

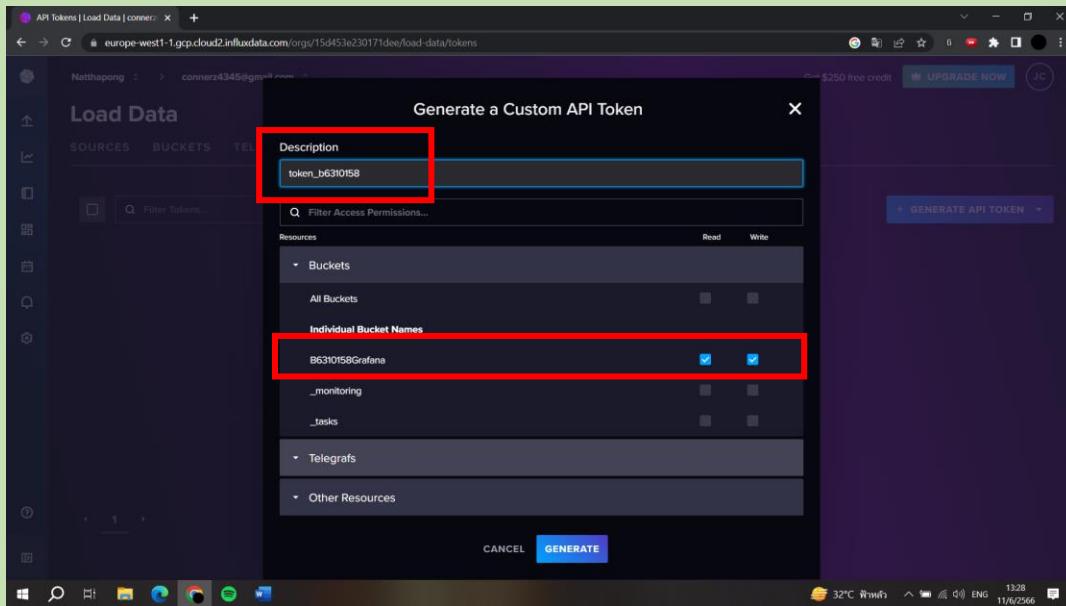
### 4. สร้าง Buckets ชื่อ B6310158Grafana

The screenshot shows the InfluxDB Cloud 'Load Data' interface. On the left, there's a sidebar with tabs for SOURCES, BUCKETS (highlighted with a red box), TELEGRAF, and API TOKENS. Below the sidebar, there are several bucket entries: 'B6310158' (Retention: 30 days, Schema Type: Implicit), '\_monitoring' (System Bucket, Retention: 7 days, Schema Type: Explicit), and '\_tasks' (System Bucket, Retention: 3 days, Schema Type: Explicit). In the center, a 'Create Bucket' dialog box is open. It has a 'Name' input field containing 'B6310158Grafana' (highlighted with a red box), a 'Delete Data' section with 'NEVER' selected, and a 'RETENTION POLICY' dropdown set to '30 days'. There's also an 'Advanced Configuration (Optional)' button. At the bottom right of the dialog are 'CANCEL' and 'CREATE' buttons. A purple sidebar on the right provides information about what a Bucket is and how to use it.

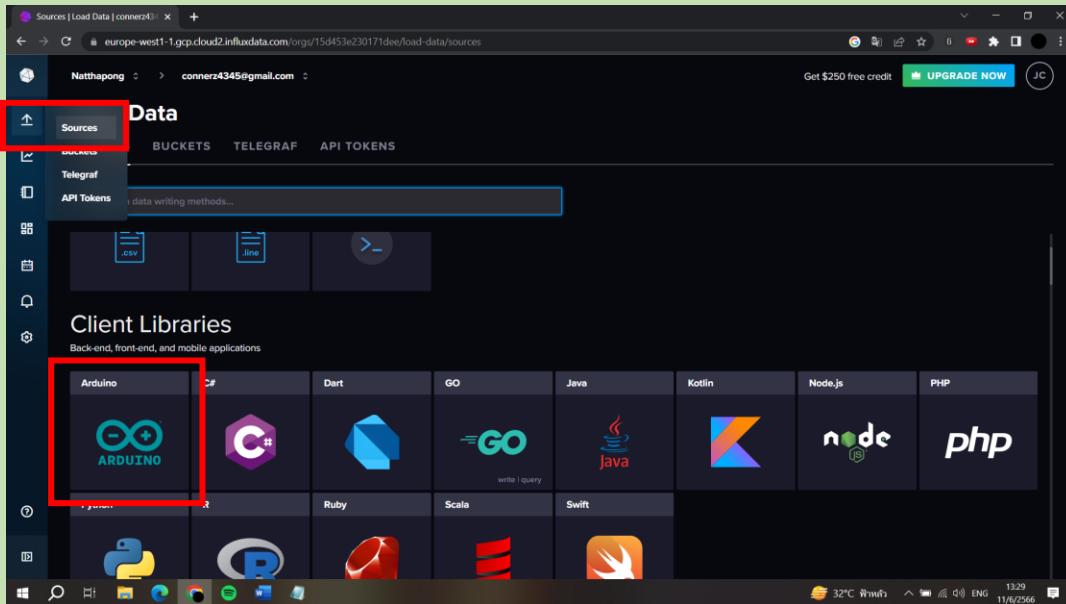
## 5. สร้าง Token



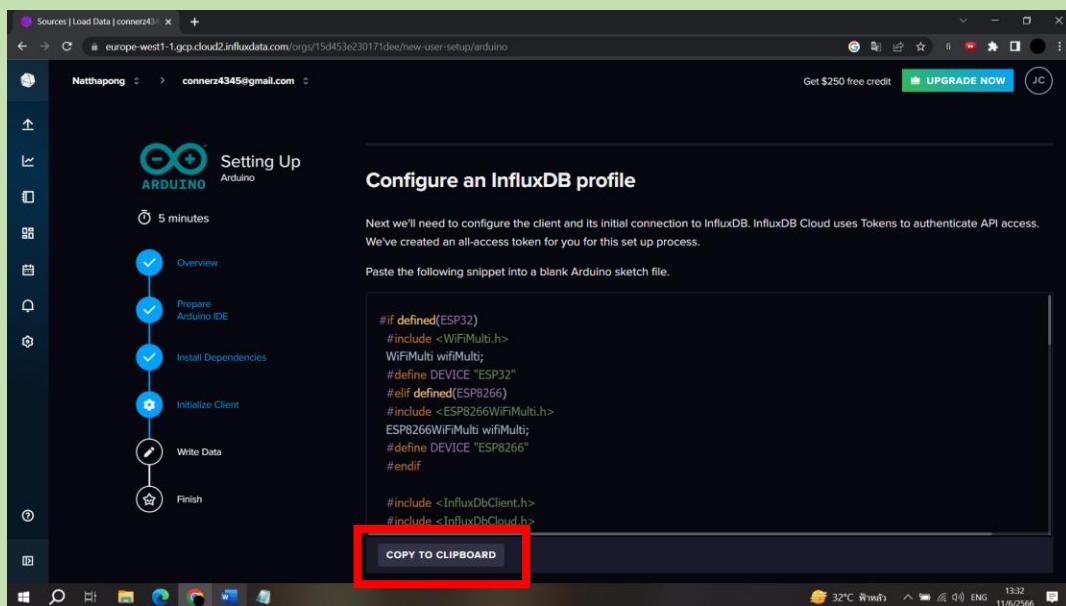
## 6. กรอกชื่อ token\_b6310158 // เลือก Bucket ที่เราสร้างขึ้นมาและคลิกเลือก Read Write



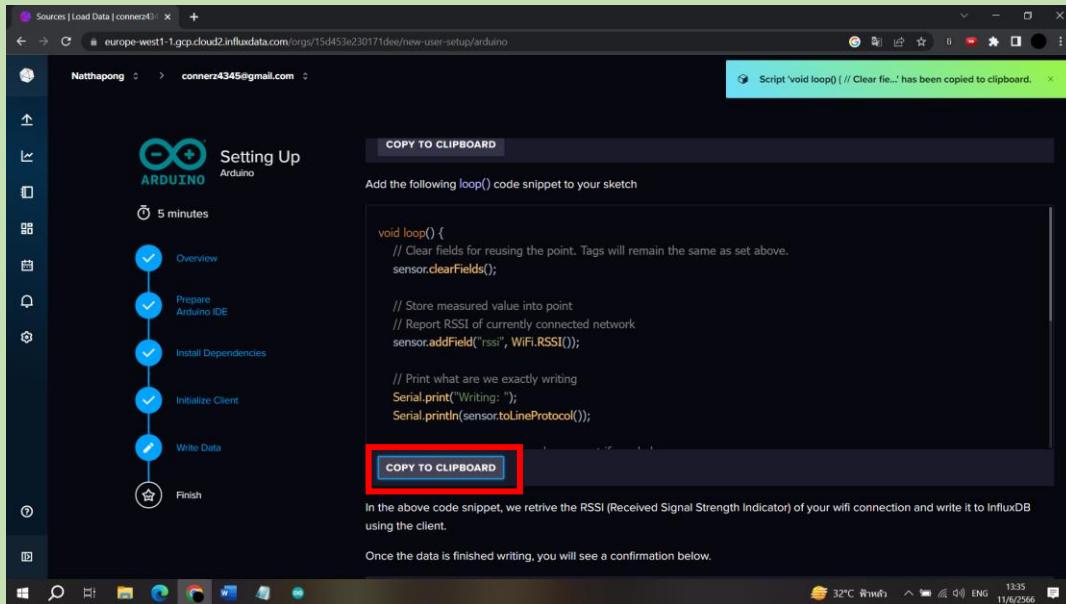
7. จากนั้นเราจะทำการสร้าง Code ในการรันบนที่ ESP32 ไปที่ Data > Sources > Client Libraries และทำการเลือก Arduino



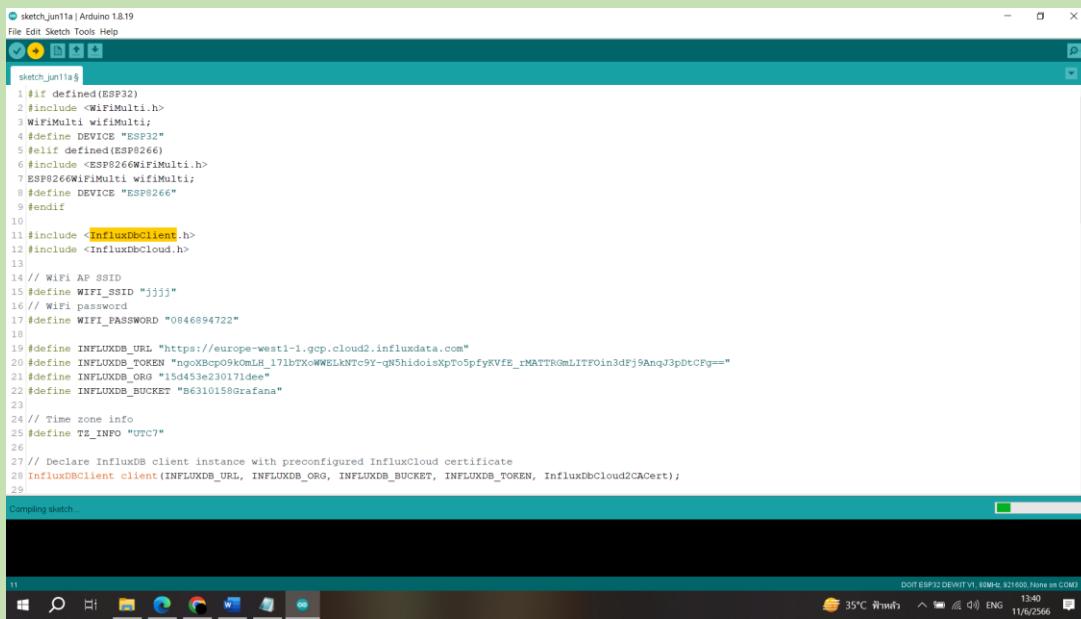
8. จากนั้นทำการ Copy to Clipboard ที่ตัว Initialize the Client



## 9. จอกนั้นทำการ Copy to Clipboard ที่ตัว Write Data



## 10. นำ Code ที่คัดลงมาวางใน Arduino



## 11. ผลการรัน

The screenshot shows the Arduino IDE interface. The left pane displays the code for 'GetChipID' (Sketch > GetChipID). The right pane shows the serial monitor window titled 'COM3' with the following log:

```
Connecting to wifi.....
Syncing time.
Synchronized time: Sun Jun 11 02:18:51 2023
Connected to InfluxDB: https://europe-west1-1.gcp.cloud2.influxdata.com
Writing: wifi_status temperature=60.67,humidity=38.02
Waiting 1 second
Writing: wifi_status temperature=27.22,humidity=43.06
Waiting 1 second
Writing: wifi_status temperature=16.81,humidity=88.39
Waiting 1 second
Writing: wifi_status temperature=32.41,humidity=87.63
Autoscroll Show timestamp Newline 115200 baud Clear output
DHT ESP32 DEWHT V1.10MHz 115200 bps on COM3
31°C 31°C ENG 11/6/2566
```

The code in the IDE is as follows:

```
#include "ESP32.h"
#include <WiFiMulti.h>
WiFiMulti WiFiMulti;
#define DEVICE "ESP32"
#include <ESP8266WiFiMulti.h>
ESP8266WiFiMulti WiFiMulti;
#define DEVICE "ESP8266"
#endif
#include <InfluxDBClient.h>
#include <InfluxDBCloud.h>
#include <Time.h>
// WiFi AP SSID
#define WIFI_SSID "jjjj"
// WiFi password
#define WIFI_PASSWORD "0846894722"
#define INFLUXDB_URL "https://europe-west1-1.gcp.cloud2.influxdata.com"
#define INFLUXDB_TOKEN "ng0XbcP09KmLH_1lbTXoWWELkNTc9Y-qN5hidoisXpTo5py"
#define INFLUXDB_ORG "15a453e230171dee"
#define INFLUXDB_BUCKET "B63101587rafana"
// Time zone info
#define TZ_INFO "UTC7"
// Declare InfluxDB client instance with preconfigured InfluxCloud certificate
#include <InfluxDBClient.h>
InfluxDBClient client(INFLUXDB_URL, INFLUXDB_ORG, INFLUXDB_BUCKET, INFLUXDB_TOKEN, InfluxDBCloud2CACert);
#include <DHT.h>
#include <DHT.h>
```

## 12. วงจรพร้อมซื้อประกอบ



## 13. Code

```

#ifndef __INFLUXDB_CLOUD_H__
#define __INFLUXDB_CLOUD_H__

#include <WiFiMulti.h>
#include <InfluxDbClient.h>
#include <InfluxDbCloud.h>

// WiFi AP SSID
#define WIFI_SSID "jjjj"
// WiFi password
#define WIFI_PASSWORD "0846894722"

#define INFLUXDB_URL "https://europe-west1-1.gcp.cloud2.influxdata.com"
#define INFLUXDB_TOKEN "npoXBcpO9kOmLH_17lbTxoWWELkNTc9Y-qN5hidoisXpTo5pfyKVfE_rMATTRGmLITFOin3dFj9AnqJ3pDtCFg=="
#define INFLUXDB_ORG "15d453e230171dee"
#define INFLUXDB_BUCKET "B6310158Grafana"

// Time zone info
#define TZ_INFO "UTC7"

// Declare InfluxDB client instance with preconfigured InfluxCloud certificate
InfluxDBClient client(INFLUXDB_URL, INFLUXDB_ORG, INFLUXDB_BUCKET, INFLUXDB_TOKEN, InfluxDbCloud2CACert);

// Declare Data point
Point sensor("wifi_status");

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

 // Setup wifi
 WiFi.mode(WIFI_STA);
 wifiMulti.addAP(WIFI_SSID, WIFI_PASSWORD);

 Serial.print("Connecting to wifi");
 while (wifiMulti.run() != WL_CONNECTED) {
 Serial.print(".");
 delay(100);
 }
 Serial.println();
}

#endif

```

```

// Accurate time is necessary for certificate validation and writing in
// batches
// We use the NTP servers in your area as provided by:
// https://www.pool.ntp.org/zone/
// Syncing progress and the time will be printed to Serial.
timeSync(TZ_INFO, "pool.ntp.org", "time.nis.gov");

// Check server connection
if (client.validateConnection()) {
 Serial.print("Connected to InfluxDB: ");
 Serial.println(client.getServerUrl());
} else {
 Serial.print("InfluxDB connection failed: ");
 Serial.println(client.getLastErrorMessage());
}
}

void loop() {
 // Clear fields for reusing the point. Tags will remain the same as set
 // above.
 sensor.clearFields();
 float temp = random(0, 6500) * 0.01 ;
 float humid = random(2500, 9000) * 0.01 ;
 // Store measured value into point
 sensor.addField("temperature", temp);
 sensor.addField("humidity", humid);

 // Print what are we exactly writing
 Serial.print("Writing: ");
 Serial.println(sensor.toLineProtocol());

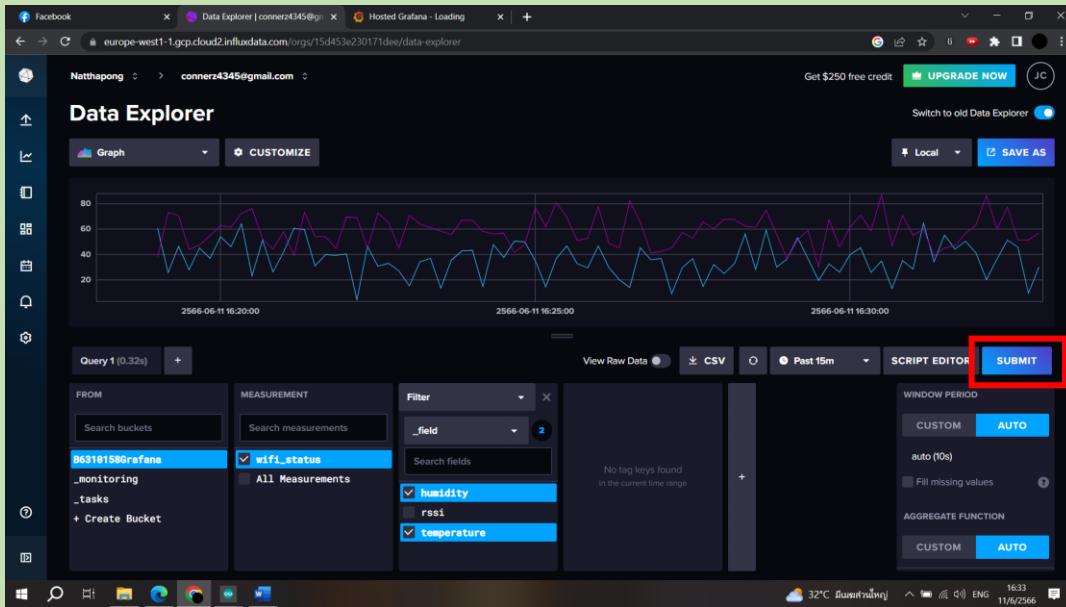
 // Check WiFi connection and reconnect if needed
 if (wifiMulti.run() != WL_CONNECTED) {
 Serial.println("Wifi connection lost");
 }

 // Write point
 if (!client.writePoint(sensor)) {
 Serial.print("InfluxDB write failed: ");
 Serial.println(client.getLastErrorMessage());
 }

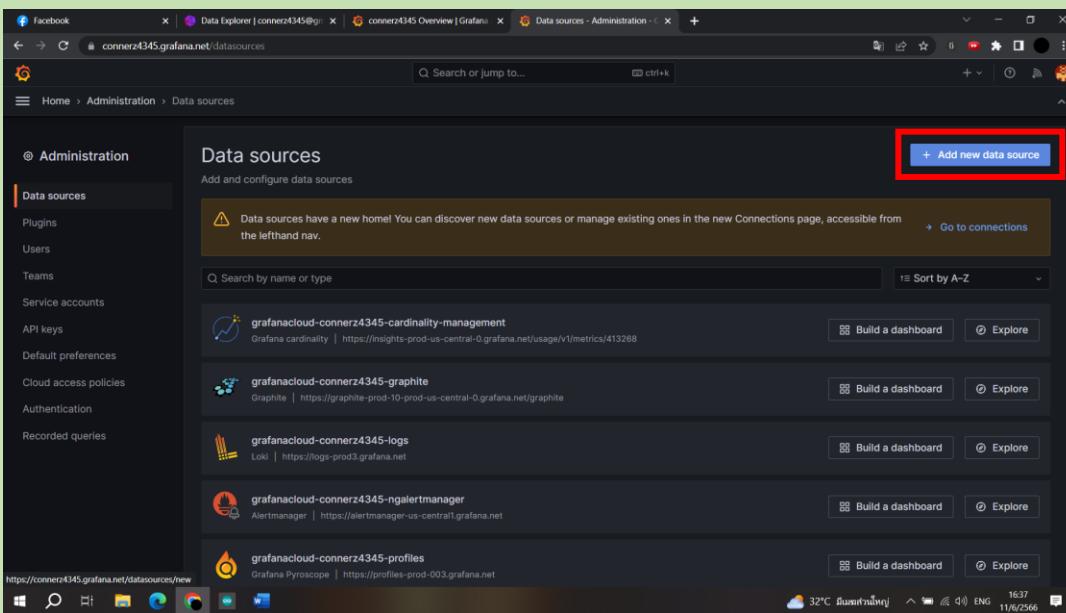
 Serial.println("Waiting 1 second");
 delay(1000);
}

```

14. 既然นั้นไปที่ Export เราจะมาดูค่าทำการ random ที่ตัว ESP32 แล้วทำการส่งค่ามาที่ตัว InfluxDB ไปที่ Export หลังจากนั้นทำการเลือกที่ต้องการเลือกดูและกด Submit



15. 既然นั้น เข้าเว็บ <https://grafana.com/> แล้วทำการ login และ ทำการสร้าง Data source ขึ้นมา ไปที่ Configuration > Data source 既然นั้นทำการ Add data source



## 16. Add InfluxDB เข้าไป

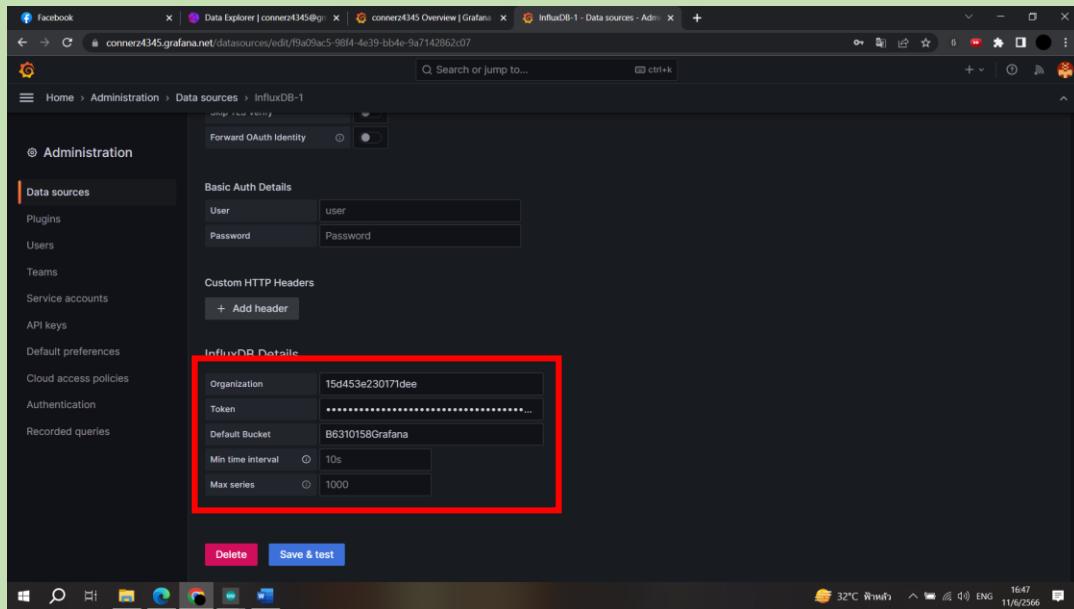
The screenshot shows the 'Add data source' interface in Grafana. On the left sidebar, under 'Data sources', 'InfluxDB' is selected. The main area displays several time series databases: Prometheus, Graphite, InfluxDB, and OpenTSDB. The 'InfluxDB' section is highlighted with a red box. It includes a brief description: 'Open source time series database & alerting' and a 'Core' button.

## 17. Query Language ให้เลือกเป็น Flux // ในส่วนของ URL นั้นเอาลิงค์ของ URL InfluxDB มา

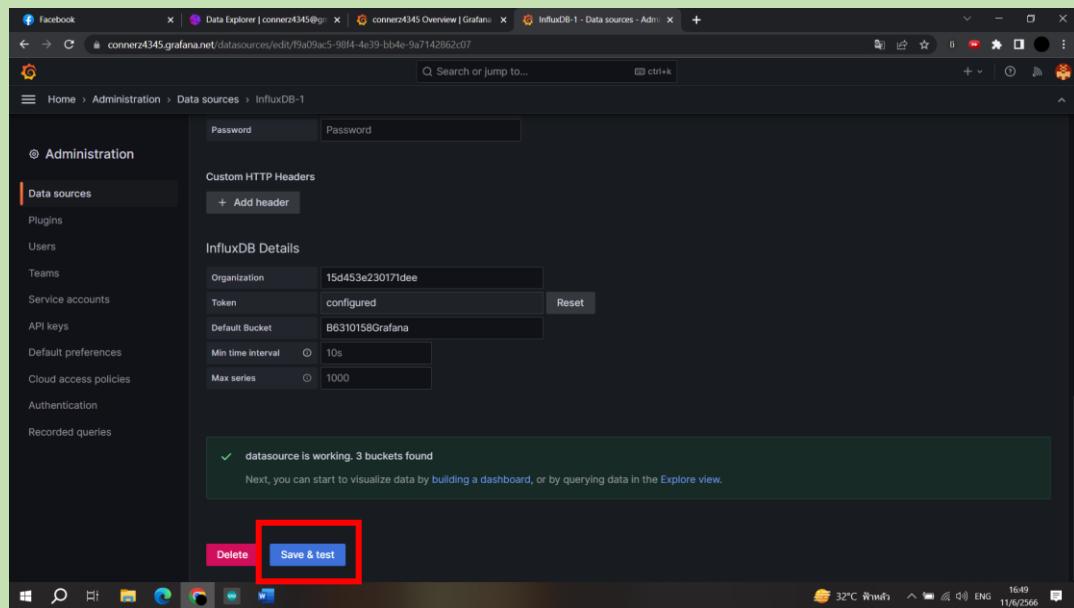
<https://europe-west1gcp.cloud2.influxdata.com/>

The screenshot shows the 'InfluxDB-1' configuration page. Under the 'Settings' tab, the 'Query Language' dropdown is set to 'Flux', which is highlighted with a red box. Below it, there is a note: 'Support for Flux in Grafana is currently in beta'. At the bottom of the page, the 'URL' field contains the value 'https://europe-west1gcp.cloud2.influxdata...'. The entire configuration page is framed by a red border.

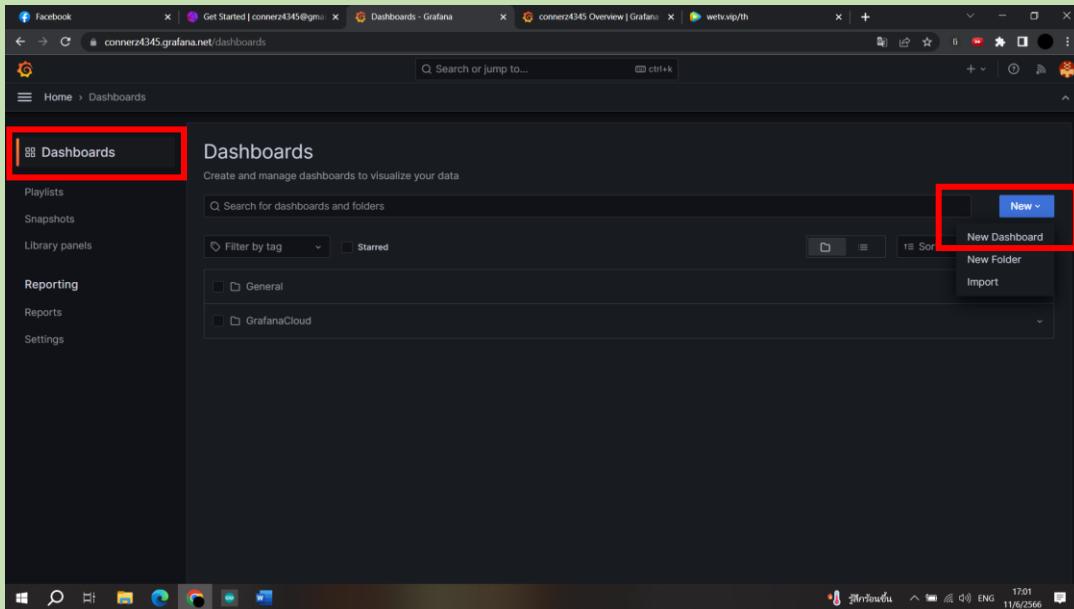
18. กรอก InfluxDB Details เป็น ORG, Token, Default Bucket ตามที่ได้สมัครไว้



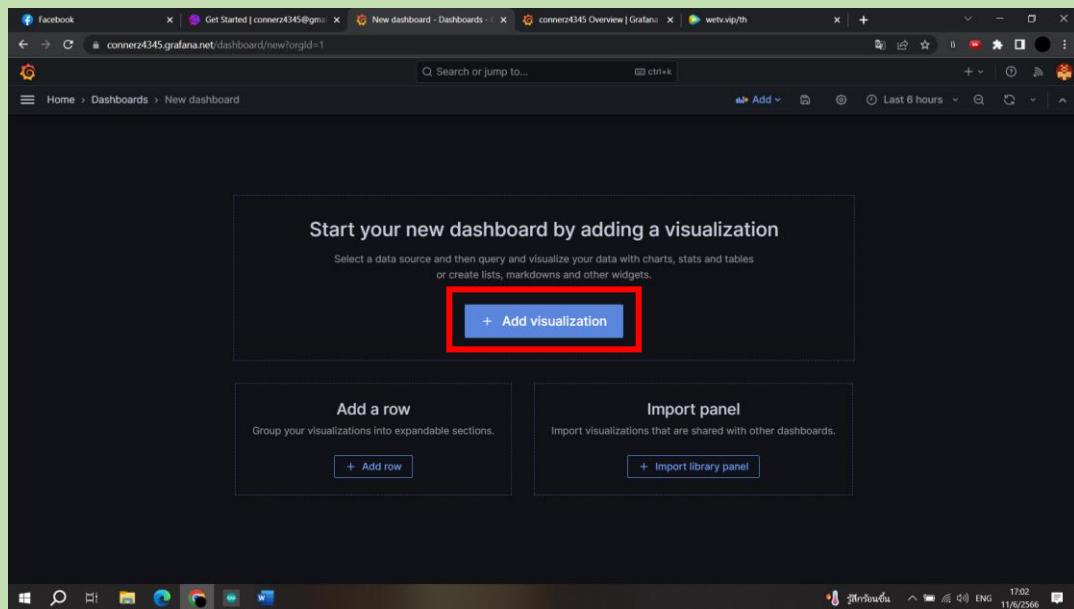
19. หากเข้ามายังหน้า database สำเร็จ เมื่อคลิก Save & test จะขึ้นจำนวน buckets ทั้งหมดใน database ที่เจอ



20. กลับมาที่เว็บไซต์ Dashboard ของ Grafana ที่ทำการตั้งค่าไว้ในข้างต้น เลือก dashboard --> new --> new dashbord



21. เว็บไซต์จะปรากฏให้เพิ่ม panel สำหรับการแสดงผลข้อมูล คลิกเลือก Add visualization



22. ในช่องของ Data source นั้นให้เลือกไปที่ InfluxDB ตามข้อที่สร้าง Data source ไว้

The screenshot shows the 'Select data source' dialog in Grafana. The 'InfluxDB-1' option is highlighted with a red box. Other options listed include 'grafanacloud-connerz4345-prom' (Prometheus), 'Grafana Cloud k6' (Grafana Cloud k6), 'InfluxDB' (InfluxDB), 'grafanacloud-connerz4345-cardinality-management' (Grafana cardinality management), 'grafanacloud-connerz4345-graphite' (Graphite), 'grafanacloud-connerz4345-logs' (Loki), 'grafanacloud-connerz4345-profiles' (Grafana Pyroscope), 'grafanacloud-connerz4345-traces' (Tempo), and 'grafanacloud-connerz4345-user-insights'.

23. กรอกโค้ดเพื่อเลือกการแสดงผลข้อมูลในส่วน Query โดยในที่นี้จะมีทั้งหมด 2 Query

Query แรก เป็นของ Temperature

The screenshot shows the Grafana query editor. It displays two queries:

```

1 from(bucket: "06310158Grafana")
 |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
 |> filter(fn: (r) => r._measurement == "wifi_status")
 |> filter(fn: (r) => r._field == "humidity" or r._field == "temperature")
 |> drop(columns: ["_measurement"])
 |> drop(columns: ["_field"])
 |> aggregateWindow(every: 60s, fn:mean)

```

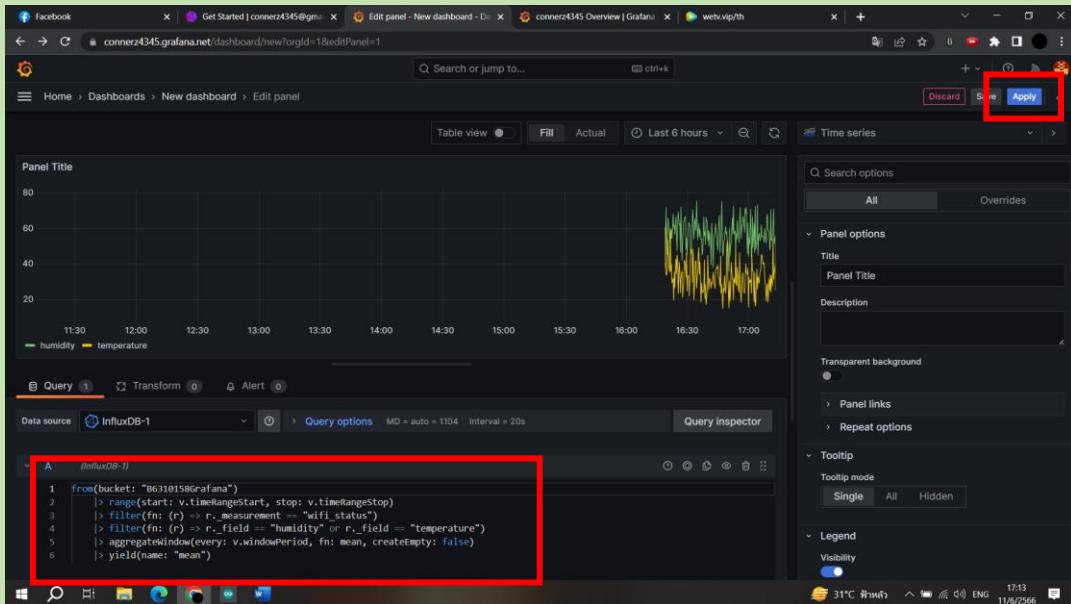
The first query is highlighted with a red box. The second query is:

```

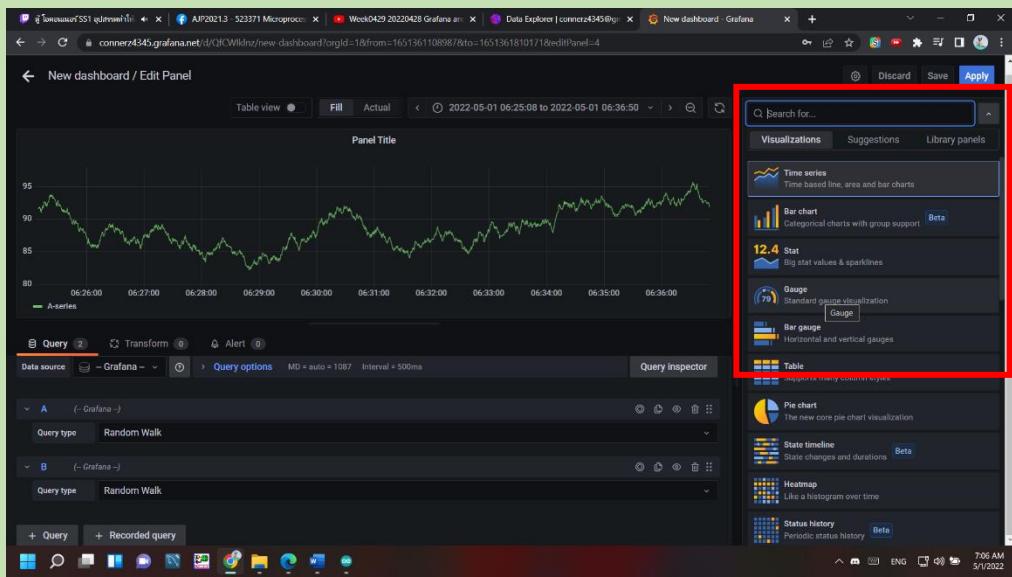
1 from(bucket: "06310158Grafana")
 |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
 |> filter(fn: (r) => r._measurement == "cpu")
 |> filter(fn: (r) => r._field == "usage_percent")
 |> drop(columns: ["_measurement"])
 |> drop(columns: ["_field"])
 |> aggregateWindow(every: 60s, fn:mean)

```

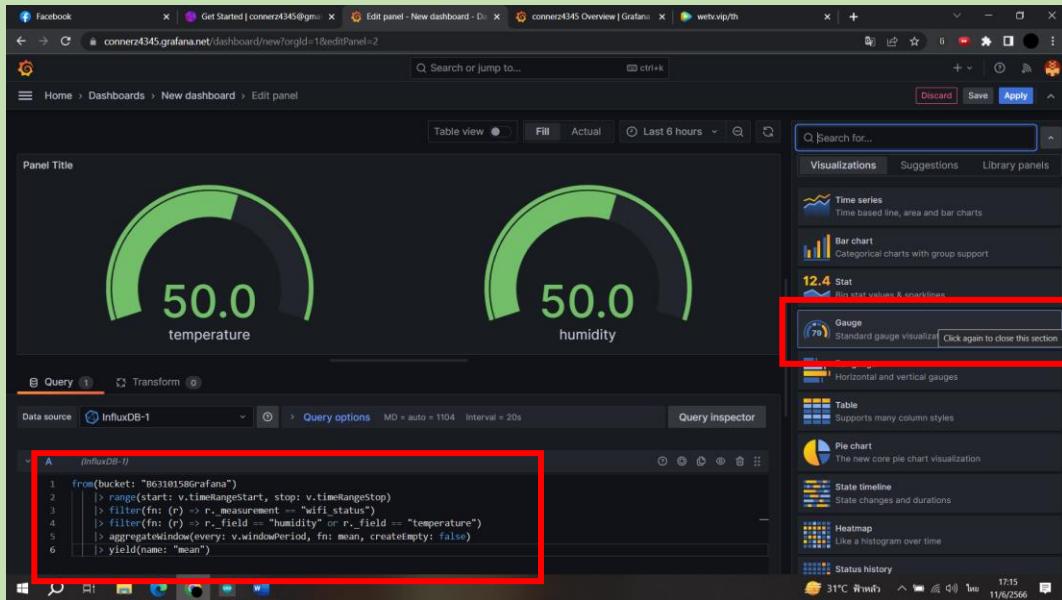
## 24. Query แรก เป็นของ Humidity แล้วกด Apply ข้างบนได้เลย



## 25. แล้วทำการสร้าง Dashboard แบบเดิมเลย เพิ่มเติมคือ ไปที่ time series แล้วเลือกที่ Gauge



## 26. ใส่ Code เพื่อนแสดงค่า Tempurature และ Humidity เหมือนเดิม



## 27. จានนักกด Apply จะได้Dashboard แสดงผล 2 panel

