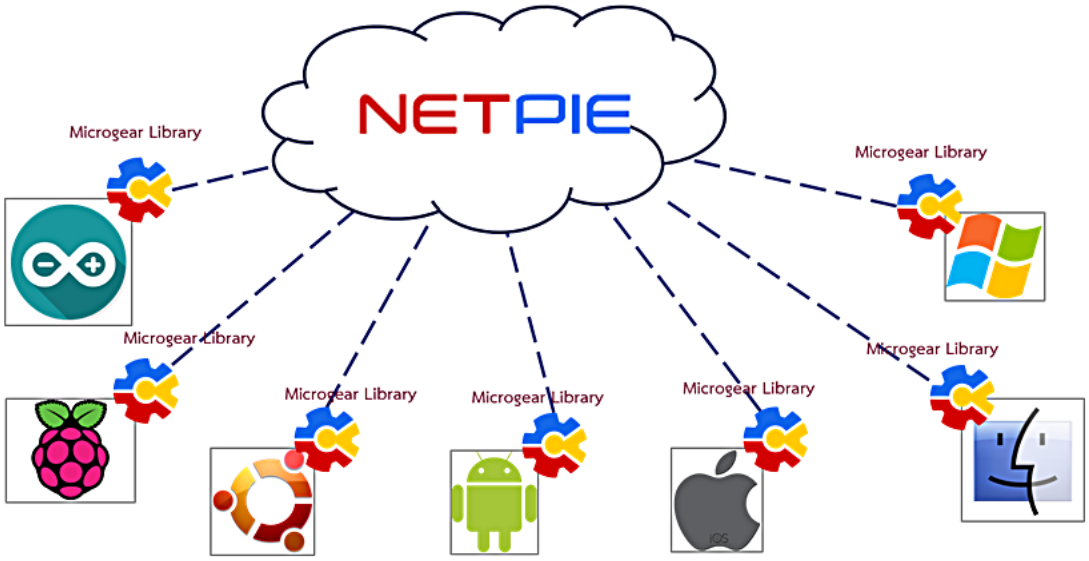
แนะนำ NETPIE

2.1. รู้จัก NETPIE

NETPIE เป็น IoT (Internet of Things) Cloud Platform ที่พัฒนาขึ้นโดยทีมงานวิจัยและเปิดให้บุคคลทั่วไปใช้งาน โดยมี Web Portal ที่ให้สามารถลงทะเบียนและจัดการตัวตนและสิทธิของแอปพลิเคชั่นและอุปกรณ์ได้ที่เว็บไซต์ https://netpie.io ตั้งแต่เดือนกันยายน 2558 เป็นต้นมา NETPIE เป็น Middleware ที่มีหัวใจหลัก (นอกเหนือจากส่วนอื่นๆ) เป็น Distributed MQTT brokers ซึ่งเป็นเสมือนจุดนัดพบให้สิ่งต่างๆ (Things) มาติดต่อสื่อสารและทำงานร่วมกันผ่านวิธีการส่งข้อความแบบ Publish/Subscribe NETPIE มีโครงสร้างสถาปัตยกรรมเป็นคลาวด์อย่างแท้จริงในทุกองค์ประกอบ ทำให้สามารถขยายตัวได้อย่างอัตโนมัติ (Auto-scale) สามารถดูแลและซ่อมแซมตัวเองได้อัตโนมัติเมื่อส่วนหนึ่งส่วนใดในระบบมีปัญหา (Self-healing, Self-recovery) โดยไม่ต้องพึ่งผู้ดูแลระบบ การบริหารจัดการระบบเป็นแบบ Plug-and-Play ไม่ต้อง Configure หรือปรับแต่ง ในฝั่งอุปกรณ์ NETPIE มี Client Library หรือที่เรียกว่า Microgear ซึ่งทำหน้าที่สร้างและดูแลช่องทางสื่อสารระหว่างอุปกรณ์กับ NETPIE รวมไปถึงรักษาความปลอดภัยในการส่งข้อมูล Microgear เป็น Open Source และสามารถดาวน์โหลดได้จาก https://github.com/netpieio โดย ณ ปัจจุบันมี Microgear สำหรับ OS และ Embedded Board หลักๆ ที่เป็นที่นิยมในหมู่นักพัฒนาเกือบทุกชนิดโมเดลการสื่อสารของ NETPIE แสดงไว้ในรูปที่ 1.1

รูปที่ 1.1 วิธีการสื่อสารของสิ่งต่างๆ ผ่าน NETPIE



ประโยชน์ของ NETPIE

1. ช่วยลดการใช้ทรัพยากรของการเชื่อมต่อ NETPIE ช่วยให้อุปกรณ์สามารถสื่อสารกันได้โดยผู้ใช้ไม่ต้องกังวลว่าอุปกรณ์นั้นจะอยู่ที่ใด เพียงแค่นำ Microgear Library ไปติดตั้งในอุปกรณ์ NETPIE จะรับหน้าที่ดูแลเชื่อมต่อให้ทั้งหมด ไม่ว่าอุปกรณ์นั้นจะอยู่ในเครือข่ายชนิดใด ลักษณะใด หรือแม้กระทั่งเคลื่อนย้ายไปอยู่ที่ใด ผู้ใช้สามารถตัดปัญหาในการเข้าถึงอุปกรณ์จากระยะไกล (Remote Access) ด้วยวิธีการแบบเดิมๆ เช่น การใช้ Fixed Public IP Address หรือการตั้ง Port Forwarding ในเราท์เตอร์และการต้องไปลงทะเบียนกับผู้ให้บริการ Dynamic DNS ซึ่งทั้งหมดล้วนมีความยุ่งยาก ลดความยืดหยุ่นของระบบ ไม่เพียงเท่านั้น NETPIE ยังช่วยให้การเริ่มต้นใช้งานเป็นไปโดยง่าย โดยออกแบบให้อุปกรณ์ถูกค้นพบและเข้าสู่บริการโดยอัตโนมัติ (Automatic Discovery, Plug-and-Play)
2. ช่วยลดภาระด้านความปลอดภัยของข้อมูล NETPIE ถูกออกแบบให้มีระดับและสิทธิในการเข้าถึงในระดับ Fine Grain กล่าวคือผู้ใช้สามารถออกแบบได้เองทั้งหมดว่า สิ่งใดมีสิทธิคุยกับสิ่งใด สิ่งใดมีสิทธิหรือไม่-เพียงใดในการอ่านหรือเขียนข้อมูล และสิทธิเหล่านี้จะมีอายุการใช้งานนานเท่าใด หรือจะถูกเพิกถอนภายใต้เงื่อนไขใด เป็นต้น
3. ยืดหยุ่นต่อการขยายระบบ NETPIE มีสถาปัตยกรรมเป็นคลาวด์เซิร์ฟเวอร์อย่างแท้จริงในทุกองค์ประกอบของระบบ ทำให้เกิดความยืดหยุ่นและคล่องตัวสูงในการขยายตัว นอกจากนี้โมดูลต่างๆ ยังถูกออกแบบให้ทำงานแยกจากกันเพื่อให้เกิดสภาวะ Loose Coupling และสื่อสารกันด้วยวิธี Asynchronous Messaging ช่วยให้แพลตฟอร์มมีความน่าเชื่อถือได้สูง นำไปใช้ซ้ำและพัฒนาต่อได้ง่าย ดังนั้นผู้พัฒนาไม่จำเป็นต้องกังวลกับการขยายตัวเพื่อรับโหลดที่เพิ่มขึ้นในระบบอีกต่อไป

2.2. MICROGEAR

Microgear คือ ซอฟต์แวร์ไลบรารี่ของ NETPIE ที่ติดตั้งอยู่บนอุปกรณ์ที่ต้องการเชื่อมต่อสื่อสารผ่านคลาวด์ของ NETPIE Microgear เปรียบเสมือนตัวกลางและผู้ช่วยในการสร้างและดูแลการเชื่อมต่อ ให้มีความเสถียร ปลอดภัย ให้การสื่อสารแลกเปลี่ยนข้อมูลระหว่างอุปกรณ์เป็นไปอย่างราบรื่น บทบาทหน้าที่ของ Microgear สามารถแบ่งออกเป็น 4 ด้านคือ

1. ด้านการสื่อสาร (Communication) Microgear จะเป็นผู้ช่วยในการสร้างการเชื่อมต่อ (Connection) ไปยังคลาวด์ของ NETPIE และคอยตรวจสอบสถานะของการเชื่อมต่อ หากการเชื่อมต่อมีปัญหา Microgear สามารถช่วยเชื่อมต่อให้ใหม่เพื่อให้การสื่อสารเป็นไปได้อย่างราบรื่น นอกจากนี้ Microgear ยังช่วยอำนวยความสะดวก ในการสร้างช่องทางการสื่อสารแบบเข้ารหัสในกรณีที่ผู้ใช้ต้องการ ส่วนการแลกเปลี่ยนข้อมูลระหว่าง Microgear และคลาวด์ของ NETPIE จะใช้โพรโทคอล MQTT ในการสื่อสาร
2. ด้านการยืนยันตัวตน (Authentication) ในขั้นตอนการสร้างการเชื่อมต่อ Microgear จะช่วยยืนยันตัวตนของอุปกรณ์กับคลาวด์ของ NETPIE โดยการพิสูจน์ตัวตน (Identity) ของอุปกรณ์จะใช้ข้อมูลประกอบกันสามส่วนคือ AppID, App Key และ Token

3. ด้านการขออนุญาตสิทธิ (Authorization) การขออนุญาตสิทธิในการสื่อสารจะเกิดขึ้นในขั้นตอนการสร้างการเชื่อมต่อ ควบคู่กับการยืนยันตัวตน คลาวด์ของ NETPIE จะเป็นผู้ออกใบอนุญาต (Token) ที่ระบุว่าอุปกรณ์ตัวนี้สามารถสื่อสารได้กับอุปกรณ์ตัวใดบ้าง ในกรณีปกติอุปกรณ์ที่อยู่ภายใต้กลุ่ม AppID เดียวกันเท่านั้น จึงจะมีสิทธิสื่อสารกันได้ (ยกเว้นในกรณีการใช้ Freeboard Microgear ที่อนุญาตให้สื่อสารข้าม AppID ได้ ซึ่งจะอธิบายในบทที่ 5)
4. ด้านการประสานงาน (Coordination) Microgear มีฟังก์ชั่นที่ช่วยให้อุปกรณ์ต่างๆ ภายในกลุ่ม AppID เดียวกันทราบสถานะของกันและกัน เช่นทราบว่ามีอุปกรณ์ใดออนไลน์เข้ามาใหม่ในกลุ่ม หรือมีอุปกรณ์ใดออกไปจากกลุ่ม รวมถึงทราบการเปลี่ยนแปลงสถานะของอุปกรณ์ที่สนใจติดตาม จากข้อมูลดังกล่าวผู้ใช้สามารถกำหนดบทบาทหน้าที่ให้อุปกรณ์ในกลุ่มตามสถานะของอุปกรณ์อื่นๆ ในกลุ่ม เช่น หากเป็นอุปกรณ์ตัวแรกในกลุ่มให้ทำหน้าที่เป็นหัวหน้ากลุ่มเป็นต้น MicrogearLibrary ถูกพัฒนาขึ้นเพื่อให้ทำงานได้กับอุปกรณ์ที่หลากหลาย ในส่วนของซอฟต์แวร์มี Microgear ให้เลือกใช้กับ Programming Language ที่หลากหลาย อาทิเช่น Node.js Python HTML5 Java (อยู่ระหว่างการทดสอบ) Android (อยู่ระหว่างการทดสอบ)สำหรับอุปกรณ์ฮาร์ดแวร์ประเภทไมโครคอนโทรลเลอร์ Microgear เปรียบเสมือน Firmware ซึ่งมี Microgear ที่รองรับ Arduino with Ethernet Shield (ใช้ได้กับ Arduino Mega) และ Microgear สำหรับ WiFi ไมโครคอนโทรลเลอร์ ESP8266 Microgear Library ที่พัฒนาขึ้นทั้งหมดถูกรวบรวมไว้ที่ https://github.com/netpieio โดยมีสัญญาอนุญาตให้ใช้สิทธิแบบเปิดประเภท ISC License ซึ่งอนุญาตให้ทำซ้ำ ดัดแปลง และ/หรือส่งต่อไลบรารี่นี้ได้ ทั้งในการใช้งานเชิงสาธารณประโยชน์และเชิงพาณิชย์ รายละเอียดของสัญญาอนุญาตให้ใช้สิทธิแบบ ISC License มีดังนี้

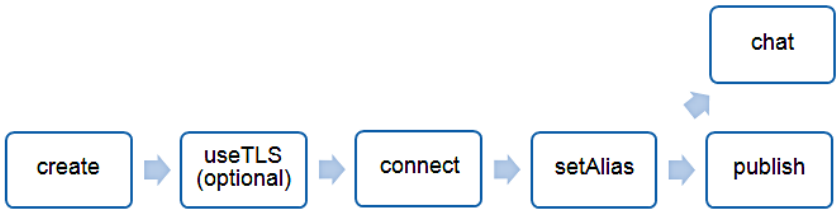
ฟังก์ชั่นหลักของ Microgear

Microgear แต่ละชนิดอาจมีชื่อและชนิดของฟังก์ชั่นแตกต่างกันตามลักษณะของการเขียนโปรแกรมในภาษานั้นๆ ในที่นี้ขอยกตัวอย่างฟังก์ชั่นที่มีใช้ทั่วไปในหลาย Microgear โดยขออ้างอิงชื่อฟังก์ชั่นจาก HTML5 Microgear สำหรับรายละเอียดฟังก์ชั่นของแต่ละชนิด Microgear สามารถดูได้จากภาคผนวก หรือเอกสาร Readme ใน https://github.com/netpie.io

- create สร้าง Microgear เพื่อเริ่มต้นใช้งาน
- connect เชื่อมต่อ Microgear เข้ากับคลาวด์ของ NETPIE
- setAlias กำหนดชื่อเล่นของอุปกรณ์เพื่อใช้ระบุตัวตนของอุปกรณ์ภายใน NETPIE
- chat ส่งข้อความแบบเจาะจงผู้รับ
- publish ส่งข้อความแบบไม่เจาะจงผู้รับไปยังหัวข้อสนทนาที่กำหนด
- subscribe ระบุความสนใจในหัวข้อสนทนา บอกรับข้อความที่เกิดขึ้นบนหัวข้อนั้นๆ
- unsubscribe ยกเลิกการบอกรับข้อความในหัวข้อสนทนาที่เคย subscribe ไว้
- resetToken ยกเลิกใบอนุญาต (Token) และลบใบอนุญาตออกจาก cache บนอุปกรณ์
- useTLS ระบุว่าต้องการสร้างการเชื่อมต่อแบบเข้ารหัสระหว่าง Microgear กับคลาวด์ของ NETPIE
- onตอบสนองต่อเหตุการณ์ที่สนใจผ่านการเรียก Callback Function

ลำดับในการเรียกฟังก์ชั่นพื้นฐานเพื่อเริ่มส่งข้อมูลเป็นไปตามแผนภาพในรูปที่ 1.2

รูปที่ 1.2 ลำดับการเรียกฟังก์ชั่นพื้นฐานของ Microgear



Events ของ Microgear

การทำงานของ Microgear เป็นแบบ Event-driven จึงต้องตอบสนองต่อเหตุการณ์ต่างๆ ด้วยการเขียน Callback Function ซึ่งชนิดของเหตุการณ์ที่สามารถเกิดขึ้น มีดังนี้

- connected เกิดขึ้นเมื่อ Microgear เชื่อมต่อกับ NETPIE สำเร็จ
- closed เกิดขึ้นเมื่อ Microgear ปิดการเชื่อมต่อกับ NETPIE
- error เกิดขึ้นเมื่อมีความผิดพลาดเกิดขึ้นกับ Microgear

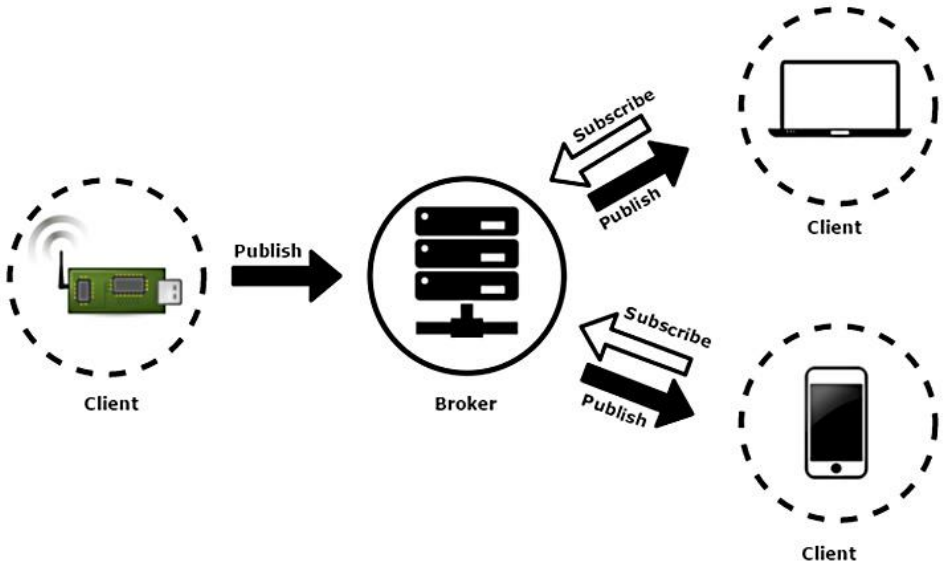
- message เกิดขึ้นเมื่อมีข้อความเข้ามาที่อุปกรณ์
- present เกิดขึ้นเมื่อมีอุปกรณ์ใน AppID เดียวกันเชื่อมต่อเข้ามาบน NETPIE
- absent เกิดขึ้นเมื่อมีอุปกรณ์ใน AppID เดียวกันหายไปจากการเชื่อมต่อกับ NETPIE

MQTT (MQ TELEMETRY TRANSPORT)

MQTT เป็นโพรโทคอลสื่อสารชั้นแอปพลิเคชั่นที่รันบน TCP/IP ถูกพัฒนาขึ้นในปี 1999 โดย IBM และ Eurotech สำหรับการมอนิเตอร์สถานะท่อส่งน้ำมันส่วนที่วางผ่านเขตทะเลทราย ด้วยการออกแบบให้เป็นการรับส่งข้อความที่มีน้ำหนักเบามากและเป็นโพรโทคอลเปิด ทำให้ในปัจจุบัน MQTT ถูกนำมาใช้แพร่หลายในการสื่อสารแบบ M2M หรือ IoT เพราะเหมาะสมกับอุปกรณ์ปลายทางที่มีขนาดเล็ก/พลังงานจำกัด หรือในการสื่อสารระยะไกลที่ต้องการการใช้งานแบนด์วิดธ์อย่างมีประสิทธิภาพ

โมเดลการสื่อสารของโพรโทคอลแสดงดังรูปที่ 1.3 ประกอบด้วย 2 ส่วนคือไคลเอนต์ และโบรกเกอร์

รูปที่ 1.3 โมเดลการสื่อสารของโพรโทคอล MQTT



โบรกเกอร์ เป็นจุดศูนย์กลางในการรับส่งข้อความระหว่างไคลเอนต์ วิธีการกำหนดเส้นทาง (Routing) กระทำผ่าน Topic โดยไคลเอนต์ Subscribe ใน Topic ที่ตนต้องการ จากนั้นโบรกเกอร์จะส่งข้อความทั้งหมดที่ถูก Publish ใน Topic นั้นๆ ไปให้ ดังนั้นไคลเอนต์จึงสื่อสารกันได้โดยไม่จำเป็นต้องรู้จักกัน ช่วยลดความเกี่ยวพันระหว่างผู้สร้างข้อมูลและผู้ใช้ข้อมูล ส่งผลให้การขยายตัวของเครือข่ายทำได้ง่าย นอกจากนี้หน้าที่ที่สำคัญอีกประการของโบรกเกอร์คือการรักษาความปลอดภัยของไคลเอนต์ (Authorization, Authentication) ซึ่งในส่วนนี้สามารถขยายเพิ่มเติม หรือนำไปเชื่อมกับกลไกความปลอดภัยของระบบหลังบ้านที่มีอยู่แล้วได้ ช่วยให้นำโบรกเกอร์เข้าไปใช้งานเป็นส่วนหนึ่งของระบบอื่นๆ ได้ ส่วน Authorization ของ NETPIE

ซึ่งจะได้กล่าวถึงต่อไปในหัวข้อที่ 1.4 ก็ถือเป็นตัวอย่างหนึ่งของการขยายเพิ่มเติมการรักษาความปลอดภัยของโบรกเกอร์ใน MQTT ปัจจุบันมีโบรกเกอร์ MQTT ที่เปิดให้ดาวน์โหลดไปใช้หรือดัดแปลงอยู่หลายรายได้แก่ Mosquitto, RabbitMQ, Erlang, VerneMQ ฯลฯ

ไคลเอนต์ จะเป็นได้ทั้ง Publisher หรือ Subscriber หรือ Publisher/Subscriber พร้อมๆ กัน และจะเป็นอุปกรณ์ใดๆ ก็ได้ที่สามารถรัน MQTT Client Library บน TCP/IP Stack การที่ MQTT ใช้โมเดล Publish/Subscribe ตรรกะส่วนใหญ่จึงไปตกอยู่ในฝั่งโบรกเกอร์ ทำให้ Library มีขนาดเล็ก ติดตั้งได้ง่าย ใช้งานได้กับอุปกรณ์ที่มีทรัพยากรจำกัด ไคลเอนต์จำเป็นต้องเปิดการเชื่อมต่อ TCP ไว้ตลอดเพื่อที่โบรกเกอร์จะสามารถผลักข้อความไปให้ได้ หากการเชื่อมต่อถูกตัดขาด โบรกเกอร์จะเก็บข้อความทั้งหมดที่เข้ามาไว้จนกว่าไคลเอนต์จะกลับมาออนไลน์อีกครั้ง

เมื่อเปรียบเทียบ MQTT กับ HTTP (REST) ที่มีสถาปัตยกรรมแบบ Request/Response จะพบว่า MQTT มีความได้เปรียบที่โบรกเกอร์สามารถผลัก (Push) ข้อความไปยังไคลเอนต์ได้ตามเหตุการณ์ (Event-driven) ในขณะที่เมื่อใช้ HTTP ฝั่งไคลเอนต์ต้องคอยโพลข้อมูลเป็นระยะๆ และต้องตั้งค่าคาบเวลาการโพลไว้ก่อนล่วงหน้า โดยแต่ละครั้งต้องมีการสร้างการเชื่อมต่อขึ้นใหม่และอาจจะไม่มีข้อมูลใหม่ใดๆ ให้อัพเดท ดังนั้นหากต้องการให้ระบบทำงานแบบ Real Time หรือใกล้เคียง ย่อมหมายถึงต้องตั้งคาบเวลาการโพลให้สั้น และความสิ้นเปลืองของการใช้ช่องสัญญาณที่ไม่จำเป็นที่ตามมา นี่จึงเป็นอีกเหตุผลสำคัญที่ทำให้ MQTT ได้รับความนิยมเหนือ REST สำหรับการใช้งานแบบ M2M นอกเหนือจากการมีน้ำหนักเบา

MQTT Topics

MQTT Topic เป็น UTF-8 String ในลักษณะเดียวกับ File Path คือสามารถจัดเป็นลำดับชั้นได้ด้วยการขั้นด้วย “/” ตัวอย่างเช่น myhome/floor-one/room-c/temperature ไคลเอนต์สามารถเลือก Publish หรือ Subscribe เฉพาะ Topic หรือ Subscribe หลาย Topic พร้อมๆ กันโดยใช้ Single-Level Wildcard (+) เช่น myhome/floor-one/+/temperature หมายถึงการขอเขียนหรือรับข้อความ temperature จากทุกๆ ห้องของ myhome/floor-one หรือ Multi-Level Wildcard (#) เช่น myhome/floor-one/# หมายถึงการขอเขียนหรือรับข้อความทั้งหมดที่มี Topic ขึ้นต้นด้วย myhome/floor-one เป็นต้น

เราสามารถกำหนด Topic อย่างไรก็ได้ โดยมีข้อยกเว้นการขึ้นต้น Topic ด้วยเครื่องหมาย “$” ซึ่งจะจำกัดไว้สำหรับการเก็บสถิติภายในของตัวโบรกเกอร์เท่านั้น ดังนั้นไคลเอนต์จะไม่สามารถ Publish หรือ Subscribe ไปยัง Topic เหล่านี้ได้ โดยทั่วไป Topic เหล่านี้จะขึ้นต้นด้วย $SYS

MQTT vs Message Queues เรามักพบการสับสนระหว่าง MQTT กับ Message Queues โดยคนจำนวนไม่น้อยเชื่อว่า MQ ใน MQTT มาจากคำว่า Message Queueในความเป็นจริงแล้ว MQ ในที่นี้มาจากชื่อรุ่นผลิตภัณฑ์ที่รองรับโพรโทคอล MQTT ของ IBM ที่เรียกว่า MQ Series และ MQTT ไม่ได้ทำงานในลักษณะ Message Queue กล่าวคือข้อความใน MQTT จะถูกส่งให้กับไคลเอนต์ทั้งหมดที่ Subscribe ใน Topic ในขณะที่ข้อความใน Message Queue สามารถถูกดึงออกไปใช้งานโดยผู้ใช้เพียงรายเดียวเท่านั้น

MQTT Connections

แพ็กเกตควบคุม (Control Packets) ทั้งหมดที่ใช้ในการสื่อสารระหว่างโบรกเกอร์กับไคลเอนต์ใน MQTT มีทั้งสิ้น 14 ชนิด แสดงไว้ดังตารางที่ 1.1

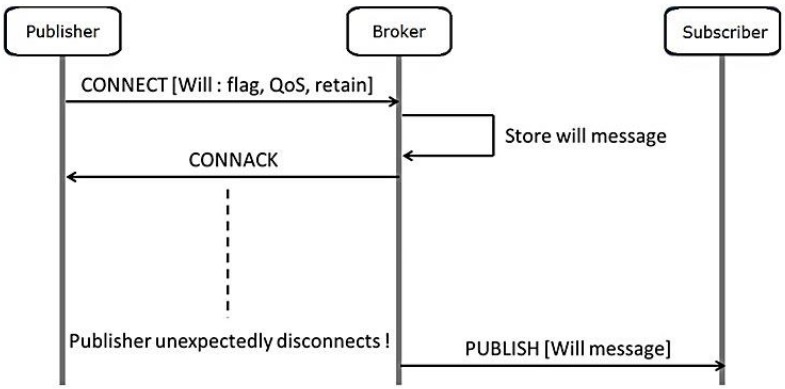
แพ็กเกตควบคุม	ผู้ส่ง(โบรกเกอร์)	ผู้ส่ง(ไคลเอนต์)	คำอธิบาย
CONNECT		X	ขอเชื่อมต่อ
CONNACK	X		รับทราบการขอเชื่อมต่อ
PUBLISH	X	X	ข้อความที่จะขอ Publish
PUBACK	X	X	แจ้งว่าได้ Publish แล้ว (QoS Level 1)
PUBREC	X	X	แจ้งว่าได้ Publish แล้ว (QoS Level 2)
PUBREL	X	X	รับทราบว่าข้อความถูก Publish แล้วและลบให้อีกผู้รับลบค่าสถานะได้ (QoS Level 2)
PUBCOM	X	X	แจ้งว่า Publish เสร็จสิ้นและสถานะถูกลบ (QoS Level 2)
SUBSCRIBE		X	ขอ Subscribe
SUBACK	X		รับทราบการขอ Subscribe
UNSUBSCRIBE		X	ขอยกเลิก Subscribe
UNSUBACK	X		รับทราบการขอยกเลิก Subscribe
PINGREQ		X	PING Request
PINGRESP	X		PING Response
DISCONNECT		X	ขอยกเลิกการเชื่อมต่อ

การเชื่อมต่อ MQTT จะเริ่มต้นจากฝั่งไคลเอนต์ส่งแพ็กเกตควบคุม CONNECT ไปยังโบรกเกอร์ โบรกเกอร์จะตอบรับด้วยแพ็กเกตควบคุม CONNACK วิธีนี้ช่วยแก้ปัญหาไคลเอนต์ที่ติดตั้งอยู่หลังเราท์เตอร์หรือไม่มีเลขไอพีสาธารณะ เพราะโบรกเกอร์มีเลขไอพีสาธารณะและจะรักษาการเชื่อมต่อสองทางไว้ตลอดหลังจากได้รับแพ็กเกตควบคุม CONNECT หากโบรกเกอร์พบว่าแพ็กเกต CONNECT ที่ได้รับไม่ถูกต้อง หรือไคลเอนต์ใช้เวลานานเกินไปนับตั้งแต่เปิดซ็อกเก็ตจนกระทั่งเริ่มส่งแพ็กเกต โบรกเกอร์จะปิดการเชื่อมต่อเพื่อป้องกันไคลเอนต์ไม่ประสงค์ดีที่ต้องการถ่วงการทำงานของโบรกเกอร์

ไคลเอนต์จะเป็นผู้ระบุค่าการเชื่อมต่อในแพ็กเกตควบคุม CONNECT ค่าเหล่านี้ ได้แก่

- Client ID เพื่อให้โบรกเกอร์ใช้ระบุตัวตนของไคลเอนต์และเก็บค่าสถานะเซสชั่นได้แก่ Subscriptions และข้อความทั้งหมดที่ไคลเอนต์ยังไม่ได้รับไว้ให้ (ส่วนนี้เกี่ยวข้องกับการตั้งค่าระดับ QoS ซึ่งจะกล่าวถึงต่อไป) ดังนั้น Client ID จึงจำเป็นที่จะต้องไม่ซ้ำกัน แต่หากไม่ต้องการให้โบรกเกอร์เก็บค่าสถานะ ไคลเอนต์สามารถเว้นส่วนนี้ว่างไว้ได้
- Clean Session มีค่า True หรือ False เพื่อระบุว่าไคลเอนต์ต้องการให้โบรกเกอร์รักษาค่าสถานะของเซสชั่นไว้ให้หรือไม่ หากต้องการ ให้ระบุค่าเป็น False หากไม่ ให้ระบุค่าเป็น True ดังนั้นหากไคลเอนต์ไม่ระบุค่า Client ID ในแพ็กเกต CONNECT จะต้องระบุ Clean Session เป็น True ด้วย มิฉะนั้นโบรกเกอร์จะไม่รับการเชื่อมต่อ
- Username และ Password เพื่อให้โบรกเกอร์ใช้ในการ Authentication และ Authorization ซึ่งตามมาตรฐานปัจจุบัน (MQTT 3.1.1) โพรโทคอล MQTT เองไม่มีการเข้ารหัสหรือแฮชในส่วนนี้ กล่าวคือ Username/Password จะถูกส่งเป็น Plaintext ดังนั้นจึงมีข้อควรระวังในการใช้ MQTT หากไม่มีชั้นความปลอดภัยเพิ่มเติมเช่น TLS ในชั้นทรานสปอร์ต
- Last Will Topic, Last Will QoS และ Last Will Message มีไว้ให้โบรกเกอร์ Publish ข้อความสั่งเสียสุดท้าย (Last Will Message) ไปยัง Topic ที่ระบุ (Last Will Topic) เพื่อให้ไคลเอนต์อื่นรับรู้ในกรณีการเชื่อมต่อของไคลเอนต์รายนี้ขาดลงแบบไม่เจตนา
- Keep Alive เป็นค่าตัวเลขคาบเวลาที่ไคลเอนต์ตกลงกับโบรกเกอร์ว่าจะส่งแพ็กเกตควบคุม PINGREQ มาเป็นระยะๆ ซึ่งโบรกเกอร์จะตอบด้วยแพ็กเกต PINGRESP เพื่อให้ทั้งสองฝ่ายรับรู้ว่าการเชื่อมต่อยังคงอยู่

รูปที่ 1.4 ผังการรับส่งข้อความสั่งเสียสุดท้าย (Last Will Message) [1]



ไคลเอนต์ Publish ข้อความ โดยบรรจุลงในส่วน Payload ของแพ็กเกตควบคุม PUBLISH ซึ่งจะต้องระบุ Packet ID, ชื่อ Topic, ระดับของ QoS, Duplicate Flag และ Retain Flag โบรกเกอร์จะตอบกลับด้วยแพ็กเกต PUBACK หรือ PUBREC ขึ้นอยู่กับระดับ QoS ที่ระบุในแพ็กเกต PUBLISH ในทางกลับกันเมื่อต้องการรับข้อมูล ไคลเอนต์ส่งแพ็กเกตควบคุม SUBSCRIBE ไปยังโบรกเกอร์ โดยระบุรายชื่อ Topic ที่ต้องการ ซึ่งอาจมีได้มากกว่าหนึ่ง และสามารถเลือกตั้งค่าระดับ QoS ที่แตกต่างกันสำหรับแต่ละ Topic และโบรกเกอร์จะตอบด้วยแพ็กเกต SUBACK โดยยืนยันค่าระดับ QoS ของแต่ละ Topic ที่ไคลเอนต์ขอ Subscribe กลับมาอีกครั้ง หาก Topic ใดที่ไม่อนุญาตหรือไม่ปรากฏบนโบรกเกอร์ โบรกเกอร์จะตอบกลับด้วยค่า 128 แทนที่ค่าระดับ QoS

เมื่อไคลเอนต์ต้องการยกเลิกการรับข้อมูล ทำได้โดยการส่งแพ็กเกตควบคุม UNSCRIBE ไปยังโบรกเกอร์ โดยระบุรายชื่อ Topic ที่ต้องการบอกเลิกในคราวเดียวกันได้มากกว่าหนึ่ง โบรกเกอร์จะยืนยันการยกเลิกด้วยแพ็กเกต UNSUBACK

เมื่อไคลเอนต์ต้องการเลิกการเชื่อมต่อ ทำได้โดยการส่งแพ็กเกตควบคุม DISCONNECT ไปยังโบรกเกอร์ หากไคลเอนต์ CONNECT โดยตั้งค่า Clean Session เป็น True โบรกเกอร์จะยกเลิก Subscription ทั้งหมดของไคลเอนต์ให้เองโดยอัตโนมัติ ในทางกลับกันหาก Clean Session เป็น False โบรกเกอร์จะยังคงเก็บค่าต่างๆ ของเซสชั่นไว้ เมื่อไคลเอนต์เชื่อมต่อเข้ามาใหม่ด้วย Client ID เดิม จึงไม่จำเป็นต้องเริ่ม Subscribe ใหม่อีกครั้ง

MQTT Quality of Service (QoS)

ไคลเอนต์จะเป็นผู้กำหนดระดับของบริการส่งและรับข้อความหรือ QoS ที่ตนต้องการในแต่ละ Topic ในแพ็กเกต PUBLISH หรือ SUBSCRIBE และโบรกเกอร์จะตอบสนองด้วย QoS ระดับเดียวกันสำหรับ Topic นั้นๆ

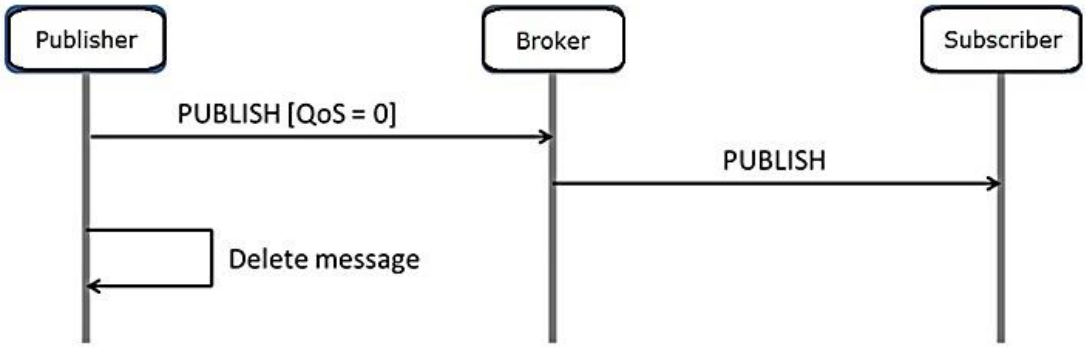
QoS ใน MQTT แบ่งได้เป็น 3 ระดับ คือ

1. อย่างมากหนึ่งครั้ง (At Most Once) แทนด้วยโค้ด 0 QoS 0 เป็นระดับบริการที่ต่ำที่สุด กล่าวคือไม่รับประกันว่าข้อความจะถูกส่งถึงผู้รับใดๆ เลยหรือไม่ หากไคลเอนต์ Publish ข้อความด้วย QoS 0 โบรกเกอร์จะไม่มีการตอบรับใดๆ ว่าได้ Publish ต่อไปให้ผู้รับรายอื่นหรือไม่ หากไม่มีผู้รับข้อความ โบรกเกอร์อาจเก็บข้อความไว้หรือลบทิ้งก็ได้ ขึ้นอยู่กับนโยบายของผู้ให้บริการเซิร์ฟเวอร์ ในทางกลับกันหากไคลเอนต์ที่เป็นผู้รับ Subscribe ไว้ด้วย QoS 0 เมื่อได้รับข้อความจากโบรกเกอร์ ก็ไม่ต้องส่งข้อความตอบรับใดๆ กลับ ทำให้การส่งข้อความแบบนี้รวดเร็วที่สุด เพราะไม่มีโอเวอร์เฮดในการตอบรับ ขณะเดียวกันหากข้อความถูกส่งไม่ถึง ก็ไม่มีทางทราบได้เช่นกัน
2. อย่างน้อยหนึ่งครั้ง (At Least Once) แทนด้วยโค้ด 1 QoS 1 รับประกันว่าข้อความจะถูกส่งถึงผู้รับอย่างน้อยหนึ่งครั้ง การส่งลักษณะนี้ ผู้ส่งจะเก็บข้อความเอาไว้ จนกว่าจะได้รับแพ็กเกต PUBACK จากผู้รับ ในกรณีไคลเอนต์ขอ Publish ผู้รับข้อความซึ่งเป็นโบรกเกอร์จะต้อง Publish ต่อไปยังไคลเอนต์ที่ Subscribe ไว้อย่างน้อยหนึ่งครั้ง จึงจะสามารถส่งแพ็กเกตตอบรับไปกลับยังผู้ส่ง ในกรณี Subscribe ผู้ส่งซึ่งก็คือโบรกเกอร์จะต้องเก็บข้อความไว้จนกว่า

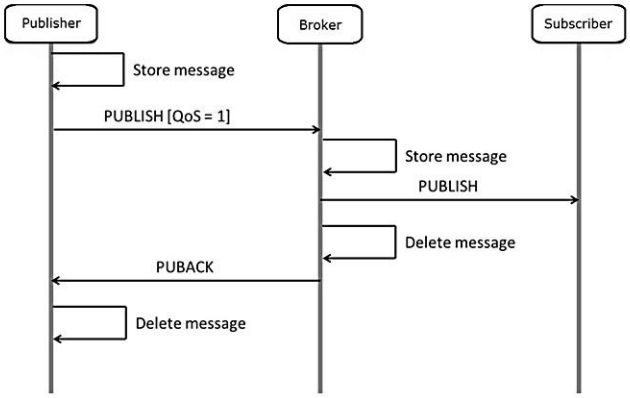
ไคลเอนต์ที่ตนส่งข้อความไปให้จะยืนยันตอบรับ ดังนั้นแพ็กเกต PUBACK จึงต้องมีหมายเลขไอดีเดียวกับแพ็กเกต PUBLISH เพื่อให้ผู้ส่งทราบว่าข้อความใดถูกส่งถึงแล้วและสามารถลบออกได้

3. หนึ่งครั้งเท่านั้น (Exactly Once) แทนด้วยโค้ด 2 QoS 2 รับประกันว่าแต่ละข้อความจะถูกส่งถึงผู้รับเพียงหนึ่งครั้งเท่านั้น เป็นบริการที่ปลอดภัยที่สุดและช้าที่สุดของโพรโทคอล MQTT เนื่องจากผู้รับและผู้ส่งต้องส่งแพ็กเกตควบคุมไปกลับถึงสองรอบ เริ่มต้นด้วยผู้ส่งส่งข้อความไปในแพ็กเกต PUBLISH เมื่อผู้รับได้รับข้อความจะเก็บแพ็กเกตไว้และยืนยันกลับไปยังผู้ส่งด้วยแพ็กเกต PUBREC ผู้ส่งจึงสามารถลบข้อความนั้นของจากหน่วยเก็บข้อมูลของตนได้ และส่งแพ็กเกต PUBREL ไปยังผู้รับเพื่อให้ผู้รับสามารถลบสถานะการส่งข้อความนี้ออกได้ หากผู้รับเป็นไคลเอนต์ปลายทางที่ Subscribe ข้อความเอาไว้ ผู้รับจะส่งแพ็กเกต PUBCOM เพื่อยืนยันว่าข้อความถูกส่งถึงแล้วเรียบร้อยหนึ่งครั้ง หากผู้รับเป็นโบรกเกอร์ ทันทีที่ได้ Publish ข้อความต่อไปยังไคลเอนต์ปลายทางหนึ่งครั้ง จึงจะลบข้อความนั้นออก และปิดเซสชั่นด้วยการส่งแพ็กเกต PUBCOM กลับไปยังผู้ส่ง

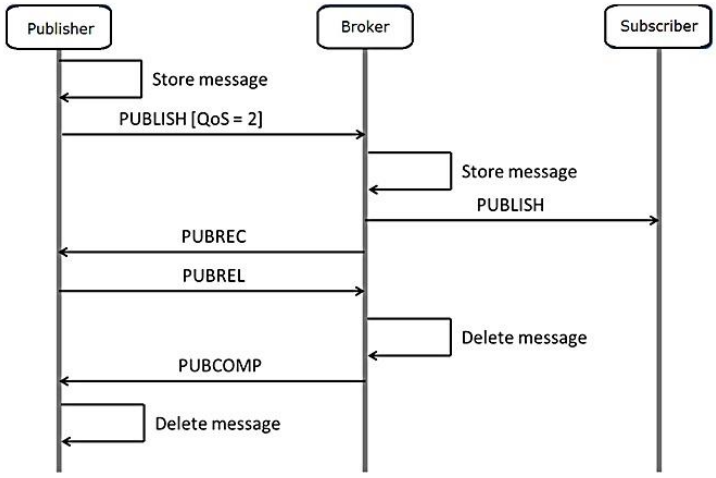
รูปที่ 1.5 ผังการสื่อสารเพื่อ Publish ข้อความด้วย QoS 0[1]



รูปที่ 1.6 ผังการสื่อสารเพื่อ Publish ข้อความด้วย QoS 1[1]



รูปที่ 1.7 ผังการสื่อสารเพื่อ Publish ข้อความด้วย QoS[1]



Retained Messages

เมื่อไคลเอนต์ Publish ข้อความแบบ Retained Message โดยการตั้ง Retain Flag เป็น True จะทำให้โบรกเกอร์เก็บข้อความนั้นไว้ใน Topic ที่ระบุอย่างถาวร จนกว่าจะมี Retained Message อื่นที่ถูก Publish ภายหลังมาเก็บแทนที่ โดยโบรกเกอร์จะเก็บ Retained Message ไว้ให้เพียง Topic ละหนึ่งข้อความ ดังนั้นทุกครั้งที่มีไคลเอนต์ Subscribe เข้ามาใหม่ ก็จะได้รับ Retained Message ทันที ไม่ต้องรอจนกว่าจะมี Publication ใหม่เกิดขึ้น จึงมองได้ว่า Retained Message คือข้อมูลสุดท้ายที่ Subscriber ทุกรายควรต้องทราบ ดังนั้น Retained Message จึงเป็นประโยชน์อย่างยิ่งกับแอปพลิเคชั่นที่เกี่ยวข้องกับการอัพเดทสถานะ หากไคลเอนต์ต้องการลบ Retained Message ใน Topic ใดๆ ก็สามารถทำได้ไม่ยาก ด้วยการส่งแพ็กเกต PUBLISH ที่มีไม่มี Payload และตั้ง Retain Flag เป็น True ไปยัง Topic นั้นๆ หรือหากมีข้อมูลจะอัพเดท ก็ไม่มีความจำเป็นต้องลบ Retained Message เก่าออกก่อน แต่สามารถส่ง Retained Message เข้าไปเขียนทับได้เลย

AUTHORIZATION และ AUTHENTICATION ใน NETPIE

เนื่องจาก NETPIE มีศูนย์กลางการสื่อสารเป็น Distributed MQTT Broker ความปลอดภัยจึงขึ้นตรงกับการเข้าถึงโบรกเกอร์ของอุปกรณ์ ดังที่ได้กล่าวไปแล้วในหัวข้อที่ 1.3ว่ามาตรฐาน MQTT ระบุให้ใช้เพียง Username และ Password ในการพิสูจน์ตัวตน (Authentication) ของไคลเอนท์ โดยสามารถกำหนดสิทธิการเข้าถึงโบรกเกอร์ของไคลเอนต์แต่ละรายด้วยการผูกเป็นรายการ (List) เข้ากับ Username บนโบรกเกอร์ ในขณะที่ NETPIE นั้น มีการแยกสิทธิออกมาจากตัวตนอย่างเด็ดขาดในรูปแบบของ Token ซึ่งจะช่วยเพิ่มความยืดหยุ่นในการบริหารจัดการการเข้าถึงโบรกเกอร์ของอุปกรณ์ให้มีความละเอียด, ยืดหยุ่นและมีประสิทธิภาพมากขึ้น ดังนั้นข้อมูลประจำตัว (Credentials) ของอุปกรณ์ที่จะมาขอเชื่อมต่อกับ NETPIE จะประกอบด้วยสองส่วนคือ Key แทนตัวตนของอุปกรณ์ และ Token ในการแทนสิทธิของอุปกรณ์

อุปกรณ์จะได้มาซึ่งข้อมูลประจำตัวจากการขออนุญาตสิทธิ (Authorization) ซึ่งประกอบด้วยผู้เกี่ยวข้อง 3 ฝ่าย คือ

1. อุปกรณ์ (Device) ที่จะมาเชื่อมต่อสื่อสารกันให้เกิดแอปพลิเคชั่น ในที่นี้รวมถึงทั้งอุปกรณ์กายภาพและอุปกรณ์เสมือน (Object)
2. ผู้ใช้ (User) ซึ่งเป็นเจ้าของแอปพลิเคชั่น และเป็นผู้กำหนดว่าอุปกรณ์ใดมีสิทธิหรือไม่ มากน้อยเพียงใดในแอปพลิเคชั่นของตน
3. NETPIE ทำหน้าที่เป็นตัวกลางในการขออนุญาตสิทธิระหว่างผู้ใช้และอุปกรณ์

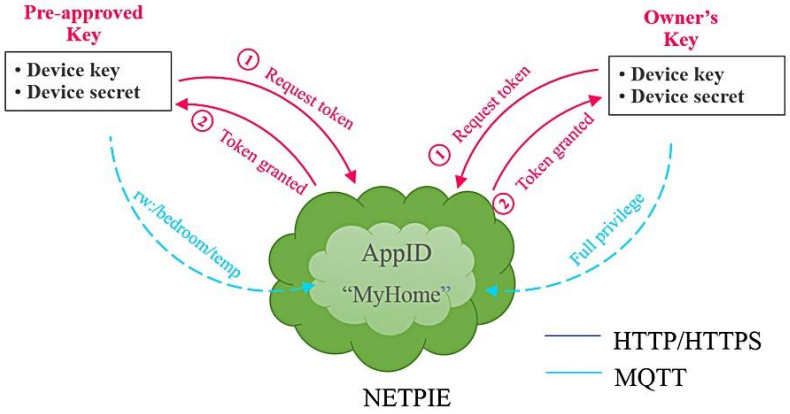
กระบวนการทั้งหมดมีขั้นตอน คือ

1. ผู้ใช้ลงทะเบียนกับ NETPIE Web Portal และสร้างแอปพลิเคชั่น
2. ผู้ใช้เพิ่มอุปกรณ์ภายใต้แอปพลิเคชั่นนั้นๆ และได้รับ Key และ Key Secret เพื่อนำมาใส่ลงในโค้ดของ Microgear ในอุปกรณ์
3. เมื่ออุปกรณ์ถูกเปิดใช้งาน Microgear จะเชื่อมต่อเข้ามายัง NETPIE โดยอัตโนมัติ และพิสูจน์ตัวตนด้วย Key และ Key Secret ที่ได้รับในการขอ Token เพื่อเชื่อมต่อและสื่อสารผ่าน NETPIE
4. ในขั้นตอนนี้ NETPIE มีทางเลือกสองทางเพื่อความสะดวกของผู้ใช้

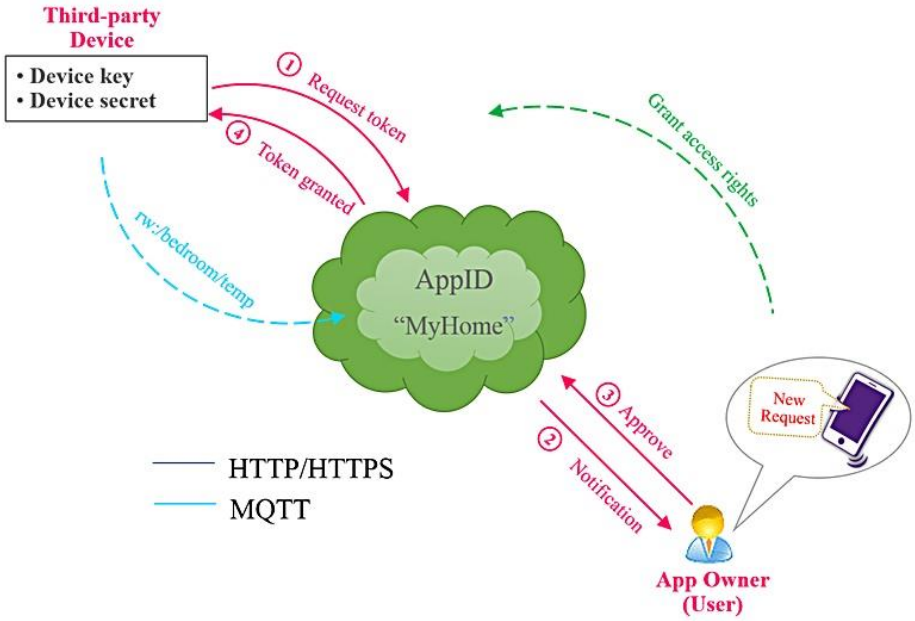
ทางเลือกที่ 1: Pre-approved Key กล่าวคือผู้ใช้สามารถกำหนดไว้ก่อนล่วงหน้าว่า Key ใดบ้างจะได้รับการอนุญาตสิทธิแบบอัตโนมัติ ดังนั้นทันทีที่อุปกรณ์ต่อเข้ามา จะได้รับ Token ทันที นอกจากนี้ผู้ใช้ยังสามารถกำหนดระดับของสิทธิไว้ก่อนได้เช่นกัน เช่น อุปกรณ์ของผู้ใช้เองให้สิทธิแบบเต็ม อุปกรณ์ของคนอื่นให้สิทธิบางส่วน เป็นต้น

ทางเลือกที่ 2: Third-Party Key ซึ่งเหมาะกับกรณีที่ผู้ใช้ให้สิทธิอุปกรณ์ของบุคคลอื่นเข้ามาร่วมใช้แอปพลิเคชั่น เมื่ออุปกรณ์ร้องขอ Token เข้ามา NETPIE AUTH Server จะติดต่อไปยังผู้ใช้ ให้เข้ามาอนุญาตสิทธิ โดยสามารถกำหนดระดับของสิทธิ และ NETPIE จะทำการออก Token ให้กับอุปกรณ์และบันทึกระดับของสิทธิที่ได้รับอนุญาตเอาไว้

รูปที่ 1.8 NETPIE Authorization ในกรณี Pre-approved Key



รูปที่ 1.9 NETPIE Authorization ในกรณี Third-party Key



เมื่ออุปกรณ์ได้ชุดข้อมูลส่วนตัวครบแล้ว จะนำทั้งหมด (Key, Key Secret, Token, Token Secret) ไปสร้าง Client ID, Username และ Password ในการพิสูจน์ตัวตนเข้าใช้ NETPIE ตามโพรโทคอล MQTT โดยวิธีการสร้างจะทำให้ Username กับ Password มีการเปลี่ยนแปลงทุกครั้งในแต่ละเซสชั่น และมีการแฮชและเข้ารหัส ช่วยป้องกันการดักฟังและการทำซ้ำ

NETPIE รองรับการเชื่อมต่อ MQTT แบบ Persistent connection เท่านั้น ดังนั้นจะมีการตั้งค่า Client ID ไว้เสมอ หากมีการเพิกถอนสิทธิของอุปกรณ์ Client ID จะถูกลบ อุปกรณ์จะต้องต่อไปยัง NETPIE เพื่อขอ Token ใหม่ โดยไม่ต้องไปเริ่มต้นกระบวนการขอ Key ใหม่ตั้งแต่ต้นกล่าวคือเพิกถอนแต่สิทธิ โดยที่ตัวตนของอุปกรณ์ยังคงอยู่ โมเดลของ NETPIE นี้เป็นการนำแนวคิดที่ใช้ในสังคมมนุษย์มาปรับใช้ในสังคม ของสิ่งของ เช่น เมื่อคนทำผิดกฎจราจร ถูกยึดใบขับขี่ ทำให้ไม่มีสิทธิในการขับขี่ยานพาหนะ แต่ตัวตนของบุคคลนั้นยังคงอยู่ และสามารถ ไปขอทำใบขับขี่ใหม่ได้ภายหลัง เป็นต้น

ชนิดของ Key

NETPIE รองรับอุปกรณ์ที่หลากหลาย นอกจากอุปกรณ์กายภาพเช่น บอร์ดฮาร์ดแวร์ต่างๆ แล้ว ยังรองรับอุปกรณ์เสมือนเช่น แอปพลิเคชั่นต่างๆ ซึ่งรวมไปถึงแอปพลิเคชั่นที่เขียนด้วย HTML5 ที่สามารถรันได้บนเบราเซอร์ ในกรณีเช่นนี้ การเชื่อมต่อไม่สามารถเป็นแบบ Persistent ได้ เนื่องจากต้องมีการปิดเปิดเบราเซอร์ตลอดเวลา ดังนั้น

เพื่อให้การจัดการ Key และ Token มีประสิทธิภาพ หลีกเลี่ยงการสร้าง Token ทิ้งไว้โดยไม่ได้ใช้งาน NETPIE จึงออกแบบให้มี Key อยู่ 2 ชนิด คือ

1. Device Key เมื่ออุปกรณ์ได้รับการติดตั้ง Device Keyและร้องขอ Token มายัง NETPIE หากได้รับการอนุญาตสิทธิ อุปกรณ์จะได้รับ Token ที่ใช้ได้ตลอดไป ไม่มีการหมดอายุ ดังนั้นตราบใดที่ไม่มีการเพิกถอนสิทธิ อุปกรณ์สามารถใช้ Token เดิมไปได้ตลอดโดยไม่ต้องเข้าสู่กระบวนการ Authorization ใหม่ Device Key จึงเหมาะกับอุปกรณ์กายภาพที่สามารถรักษาช่องการเชื่อมต่อกับ NETPIE ไว้ได้โดยตลอด
2. Session Key อุปกรณ์ที่ได้รับการติดตั้ง Session Key จะได้รับ Token ที่ใช้ได้เพียงครั้งเดียว (One-time Token) หากอุปกรณ์ยกเลิกการเชื่อมต่อไป และต่อกลับมาใหม่ จะต้องมีการนำ Session Key ไปขอ Token ใหม่ทุกครั้ง Session Key จึงเหมาะกับการใช้งานแอปพลิเคชั่นที่รันบนเบราเซอร์ที่ต้องมีการปิดเปิดอยู่เรื่อยๆ เพราะหากใช้ Device Key ทุกครั้งที่เบราเซอร์ถูกเปิดขึ้นจะมีการร้องขอ Token ใหม่ และเมื่อเบราเซอร์ถูกปิดไป Token ที่ได้จะค้างอยู่เช่นนั้นโดยไม่ได้ใช้งานอีก ส่งผลให้ภายใต้ Key เดียวกัน มี Token ทั้งที่ใช้งานและไม่ได้ใช้งานแล้วผูกอยู่เป็นจำนวนมาก สร้างความสับสนซับซ้อนที่ไม่จำเป็นให้กับผู้ใช้ จึงขอเน้นย้ำเป็นอย่างยิ่งว่า**ไม่ควรใช้ Device Key กับแอปพลิเคชั่นที่รันบนเบราเซอร์ (HTML5) โดยเด็ดขาด**

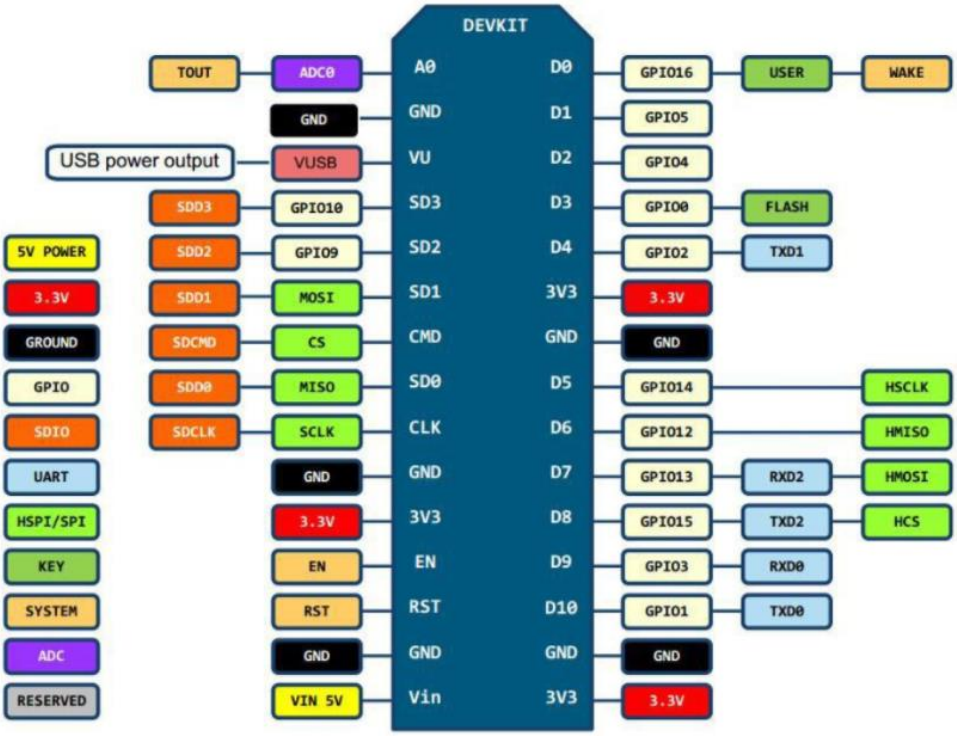
ขอบเขตของสิทธิ (Scope) ในการเข้าถึง NETPIE MQTT Broker

เมื่อผู้ใช้อนุญาตสิทธิให้กับอุปกรณ์ จะสามารถระบุขอบเขตของสิทธิ หรือที่เราเรียกว่า Scope ได้ว่ามากน้อยเพียงใด โดย Scope นี้จะผูกอยู่กับ Token ที่อุปกรณ์จะได้รับ และถูกเก็บไว้ที่ NETPIE รูปแบบของ ScopeIdentifier เกี่ยวพันโดยตรงกับโครงสร้างการสื่อสารแบบ Publish/Subscribe ไปยัง Topic บนโบรกเกอร์ MQTT มีได้ 3 ลักษณะ คือ

1. rw:/topic เช่น rw:/home/bedroom/temperature หมายถึง มีสิทธิอ่าน (Subscribe) และเขียน (Publish) ใน home/bedroom/temperature
2. r:/topic เช่น r:/home/# หมายถึง มีสิทธิอ่าน (Subscribe) ทุก Topic ที่ขึ้นต้นด้วย home
3. w:/topic เช่น w:/home/+/temperature หมายถึง มีสิทธิเขียน (Publish) ไปยัง Topic ชื่อ temperature ของทุกๆ ห้องใน home

การกำหนด Scope ที่เกี่ยวข้องกับหลาย Topic นอกจากใช้ Wildcard ดังที่ได้แสดงไปแล้ว สามารถทำได้โดยการเรียง Scope Identifier เข้าด้วยกันแล้วคั่นด้วยเครื่องหมาย Bar "|" ตัวอย่างเช่น
rw:/home/bedroom/temperature|r:/home/#|w:/home/+/temperature

ESP8266


DEVKIT
TOUT
ADC0
A0
D0
GPIO16
USER
WAKE
GND
GND
D1
GPIO5
USB power output
VUSB
VU
D2
GPIO4
SDD3
GPIO10
SD3
D3
GPIO0
FLASH
5V POWER
SDD2
GPIO9
SD2
D4
GPIO2
TXD1
3.3V
SDD1
MOSI
SD1
3V3
3.3V
GROUND
SDCMD
CS
CMD
GND
GND
GPIO
SDD0
MISO
SD0
D5
GPIO14
HSCLK
SDIO
SDCLK
SCLK
CLK
D6
GPIO12
HMISO
UART
GND
GND
D7
GPIO13
RXD2
HMOSI
HSPI/SPI
3.3V
3V3
D8
GPIO15
TXD2
HCS
KEY
EN
EN
D9
GPIO3
RXD0
SYSTEM
RST
RST
D10
GPIO1
TXD0
ADC
GND
GND
GND
GND
RESERVED
VIN 5V
Vin
3V3
3.3V

เปิด/ปิด LED ผ่าน NETPIE

การควบคุมอุปกรณ์ด้วย NETPIE Freeboard

Lab 5.2 แสดงการประยุกต์ NETPIE Freeboard ในการควบคุมอุปกรณ์ โดยในเบื้องต้นนี้ เราจะควบคุมไฟ LED บนบอร์ด NodeMCU ซึ่งใช้หลักการทางานบนพื้นฐานของการ Subscribe ข้อความจาก Topic หรือหัวข้อที่ระบุ และการกาหนดตรรกะของการควบคุมทั้งในส่วนของ Datasource และส่วนของ Widget ที่ใช้ควบคุม โดยมีขั้นตอนดังต่อไปนี้

1.แก้ไขไฟล์ pieled2.ino โดยระบุข้อมูลการเข้าถึงเครือข่าย Wifi ข้อมูล APPID, KEY และ SECRET ตามโค้ดข้างล่าง และทาการ Upload ไฟล์เข้า NodeMCUให้เชื่อมต่อกับ NETPIE

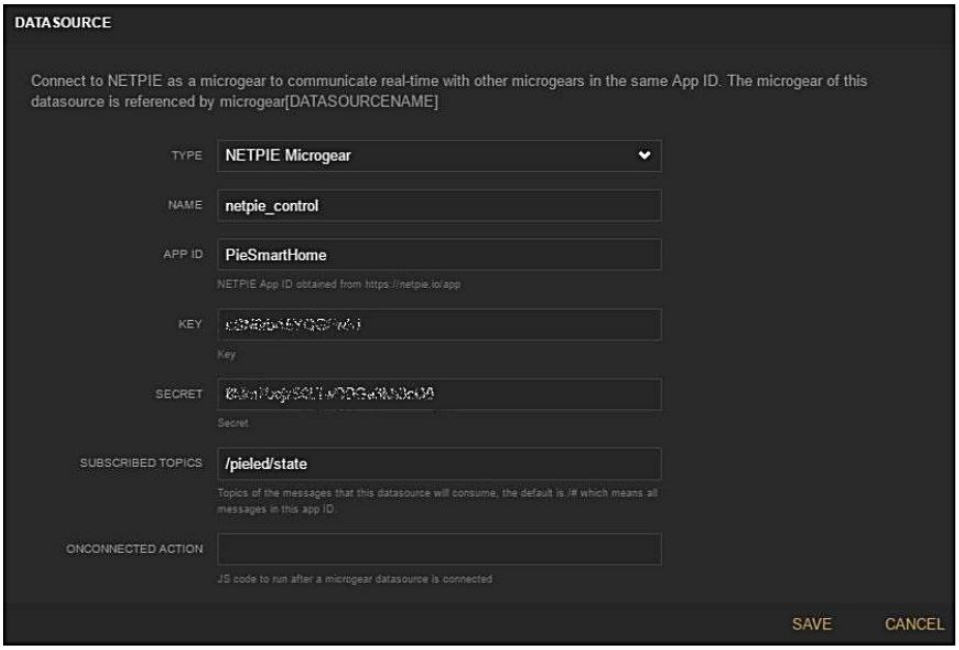
pieled2.ino

```
#include <ESP8266WiFi.h>
#include <MicroGear.h>

const char* ssid     = "SSID";
const char* password = "PASSWORD";

#define APPID   "YOUR_APPID"
#define KEY     "YOUR_KEY"
#define SECRET  "YOUR_SECRET"

#define ALIAS   "pieled"

WiFiClient client;

char state = 0;
char stateOutdated = 0;
char buff[16];

MicroGear microgear(client);

void sendState(){
  if (state==0)
    microgear.publish("/pieled/state","0");
  else
    microgear.publish("/pieled/state","1");
  Serial.println("send state..");
  stateOutdated = 0;
}

void updateIO(){
  if (state >= 1) {
    digitalWrite(LED_BUILTIN, LOW);
  }
  else {
    state = 0;
```

```
    digitalWrite(LED_BUILTIN, HIGH);
  }
}

void onMsghandler(char *topic, uint8_t* msg, unsigned int msglen) {
  char m = *(char *)msg;

  Serial.print("Incoming message -->");
  msg[msglen] = '\0';
  Serial.println((char *)msg);

  if (m == '0' || m == '1') {
    state = m=='0'?0:1;
updateIO();
  }
  if (m == '0' || m == '1' || m == '?') stateOutdated = 1;
}

void onConnected(char *attribute, uint8_t* msg, unsigned int msglen) {
  Serial.println("Connected to NETPIE...");
  microgear.setAlias(ALIAS);
  stateOutdated = 1;
}

void setup(){
    Serial.begin(115200);
    Serial.println("Starting...");

    pinMode(LED_BUILTIN, OUTPUT);

    if (WiFi.begin(ssid, password)) {
        while (WiFi.status() != WL_CONNECTED) {
            delay(500);
            Serial.print(".");
        }
    }

    microgear.on(MESSAGE,onMsghandler);
    microgear.on(CONNECTED,onConnected);
    microgear.init(KEY,SECRET,ALIAS);
    microgear.connect(APPID);
}

void loop(){
  if (microgear.connected()) {
    if (stateOutdated) sendState();
    microgear.loop();
  }
  else {
    Serial.println("connection lost, reconnect...");
    microgear.connect(APPID);
  }
}
```

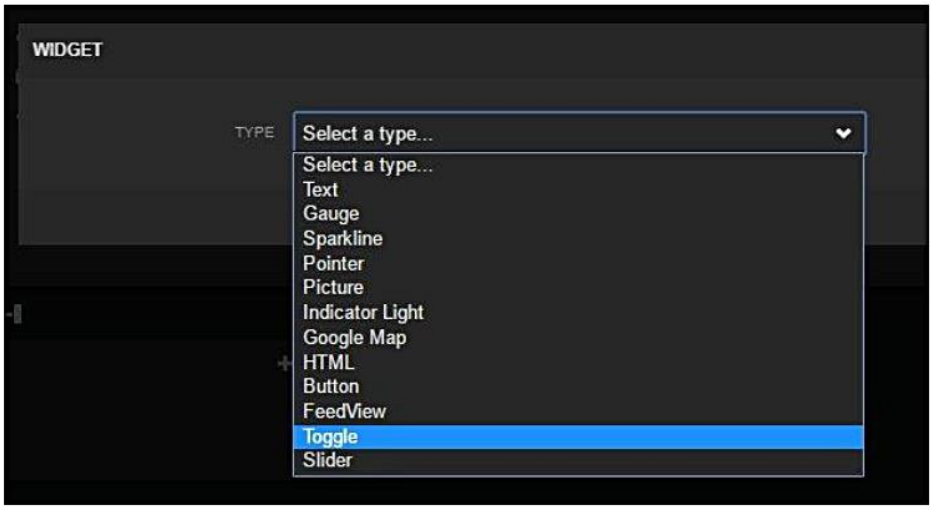
2.ในหน้า NETPIE Freeboard คลิกเพิ่ม Datasource ที่สร้างขึ้นก่อนหน้านี้เพื่อแก้ไข ตั้งชื่อ Datasource ใส่ค่า KEY และ SECRET และในช่อง SUBSCRIBED TOPICS ให้ใส่ /pieled/state หรือ Topic ที่ท่านระบุไว้สาหรับการ publish ในไฟล์ pieled2.ino ดังแสดงในภาพด้านล่าง และกด Save

รูปแสดงหน้าต่างตั้งค่า Datasource ใน NETPIE Freeboard สาหรับควบคุม LED



3.สร้าง Widget ขึ้นมาใหม่โดยกด + (ADD PANE) และเลือกชนิดใน Drop Down Box เป็นแบบ Toggle

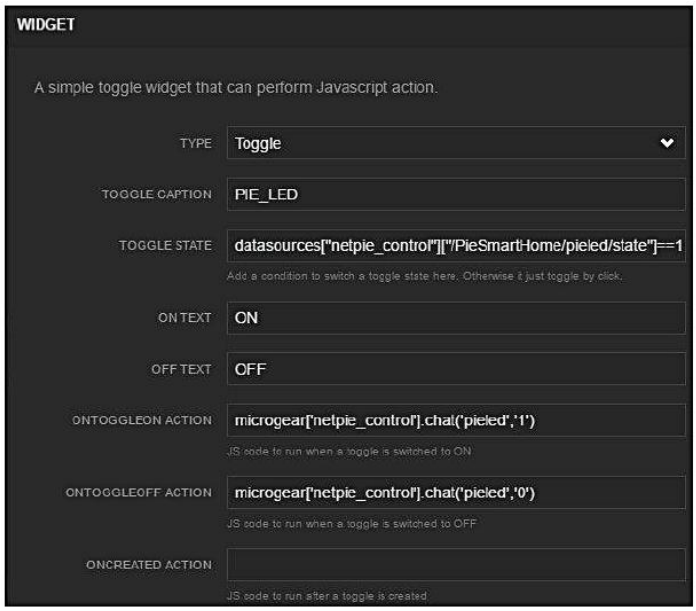
รูปแสดงหน้าจอเลือกชนิด Widget ให้เป็นแบบ Toggle



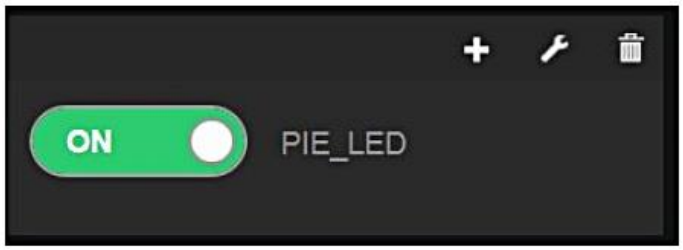
จากนั้นตั้งค่า Widget ดังนี้โดยหน้าจอการตั้งค่าแสดงดังรูป

- TOGGLE CAPTION : ตั้งชื่อปุ่ม Toggle (ในตัวอย่างตั้งเป็น PIE_LED)
- TOGGLE STATE : ใส่ข้อมูลตามชื่อของ Datasource และ Topic เช่น datasources["netpie_control"]["/PieSmartHome/pieled/state"]==1
- ON TEXT : ON
- OFF TEXT : OFF
- ONTOGGLEON ACTION : microgear['netpie_control'].chat('pieled','1')
- ONTOGGLEOFF ACTION : microgear['netpie_control'].chat('pieled','0')

รูปแสดงหน้าจอการตั้งค่า Widget ชนิด Toggle



แล้วกด Save จะได้ Widget ที่มีปุ่มควบคุมดังแสดงในภาพ เพื่อทดสอบเปิดปิด LED บน NodeMCU



คำอธิบายเพิ่มเติม

TOGGLE STATE เป็นสถานะ On/Off ซึ่งสามารถผูกกับตรรกะของ Datasource ในที่นี่เราตั้งค่าให้ Toggle เปลี่ยนสถานะตามค่าที่ส่งมาใน Topic ชื่อ /pieled/state

ONTOGGLEON และ ONTOGGLEOFF เป็นคาสั่งที่จะถูกเรียก เมื่อ Toggle เปลี่ยนสถานะไปเป็น ON และ OFF ตามลาดับ

ในหน้า Datasources ตรงช่อง SUBSCRIBED TOPICS นั้น นอกจากจะสามารถระบุค่าแบบเจาะจงเป็น topic /pieled/state แล้ว ยังสามารถระบุค่าเป็น /pieled/+ ก็ได้ โดยใช้เครื่องหมาย (+) ซึ่งเป็น Single-Level Wildcard เพื่อรับค่าของ State

เราสามารถใช้ Wildcard เพื่อช่วยในการ Subscribe Topic ต่างๆ เช่น หากต้องการ Subscribe Topic ตามที่ระบุแบบเจาะจงดังนี้: "/home/kitchen/temp", "/home/bedroom/temp", และ "/home/livingroom/temp" ก็สามารถยุบเหลือ 1 Topic คือ "/home/+/temp" นอกจาก + แล้วยังใช้เครื่องหมาย # ได้ด้วย โดยที่เครื่องหมาย + จะแทนคาอะไรก็ได้ระดับชั้นเดียว ส่วน # จะแทนคาอะไรก็ได้ในระดับชั้นยาวต่อไปเท่าไหร่ก็ได้ เช่นจะให้ match 3 Topic ข้างต้น ก็อาจจะเขียนระบุเป็น Topic คือ "/home/#"