

การใช้งาน ThingsBoard IoTs Platform เพื่อสร้างและจัดการระบบอัจฉริยะ ThingsBoard IoTs Platform for smart system

4/4 – ThingsBoard IoT Platform – Create Project

- Top 10 IoTs Platform Enterprise Scale and Startup Scale
- ThingsBoard Rule Chains and Alarm
- Case Study 1 – ตัวอย่างการสร้างระบบตรวจสอบและควบคุมอุปกรณ์
- Case Study 2 – ตัวอย่างการสร้างระบบตรวจสอบการใช้พลังงาน
- คำถามท้ายบทเพื่อทดสอบความเข้าใจ

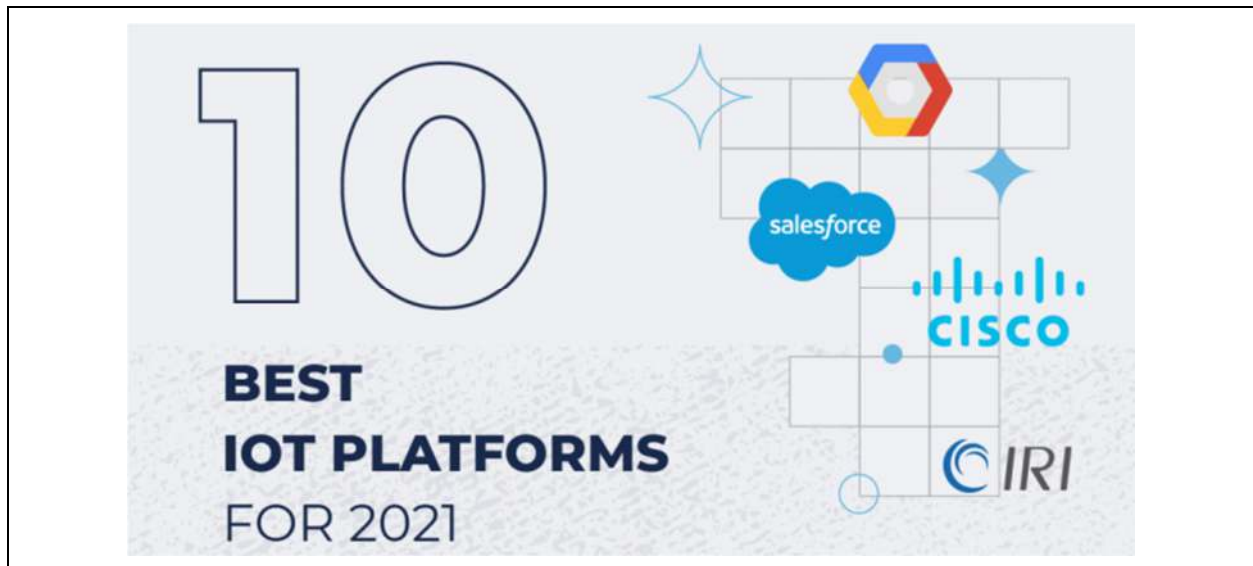
1/5 -- Top 10 IoTs Platform

1.1 Top 10 Best IoT Platforms for 2021 – Enterprise Scale

<https://www.sam-solutions.com/blog/top-iot-platforms/>

<https://www.softwaretestinghelp.com/best-iot-platforms/>

<https://keyua.org/blog/10-best-internet-of-things-iot-cloud-platforms/>

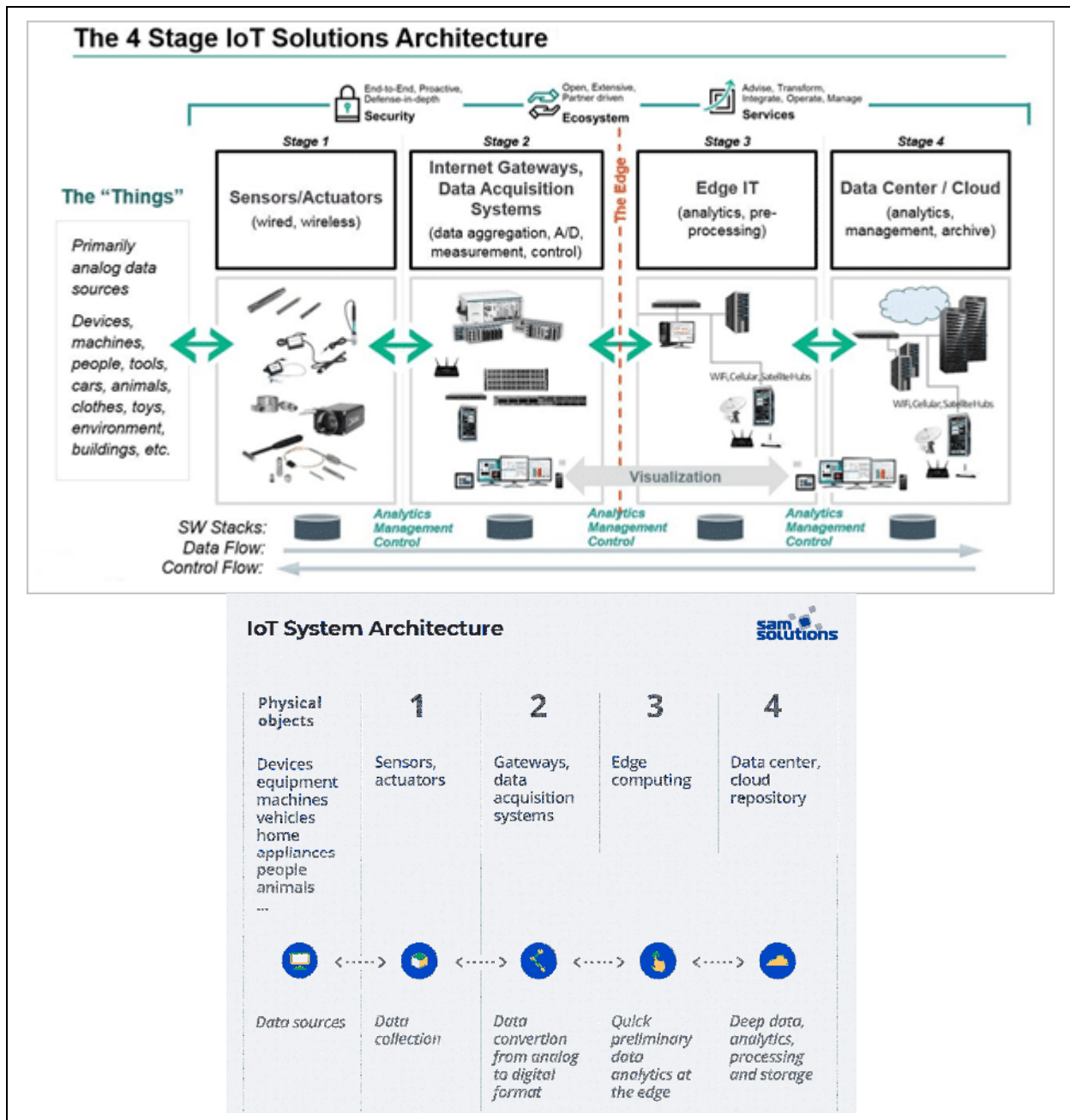


Businesses have been adopting Internet of Things technologies more actively, so the need for high-quality IoT platforms is also increasing. In this article, we discuss the most popular Internet of Things platforms for 2021.

1.1.1 IoT Technology Overview

Let's briefly look at what the Internet of Things actually is.

The Internet of Things concept implies the creation of a distributed network consisting of numerous physical objects equipped with embedded software, sensors and connectivity options that collect and share data with each other and with the central platform via the internet.



Architecture of IoT systems

IoT system architecture consists of four layers:

1. Sensors and actuators collect data directly from physical objects (devices, equipment, machines, vehicles, home appliances, people, animals, etc.).
2. Gateways and data acquisition systems convert gathered data from the analog to the digital format.
3. Edge computing ensures there's immediate preliminary data analytics right on devices.
4. Data centers or cloud services provide deep data analysis, processing and storage.

Examples of IoT systems:

- Smart home systems (security devices, intelligent lighting, conditioning, heating, connected home appliances)
- Wearable health devices both for self-tracking of health conditions (pulse oximeters, glucometers) and for vital sign monitoring in clinics
- Logistics tracking systems (GPS trackers, fuel level sensors, alert systems to monitor driver behavior)
- Autonomous vehicles (farming equipment, warehouse autonomous robots, passenger buses)
- Smart factory equipment (robotics, predictive maintenance solutions)

1.1.2 What Is an IoT Platform?

An IoT platform serves as a mediator between the world of physical objects and the world of actionable insights. Combining numerous tools and functionalities, Internet of Things platforms enable you to build unique hardware and software products for collecting, storing, analyzing and managing the plethora of data generated by your connected devices and assets.

1.1.3 Types of Internet of Things Platforms

IoT products consist of numerous components:

- Hardware
- Software
- Communication technologies
- Central repository (cloud or local)
- End-user applications

To cover each aspect while developing an IoT product, there are several types of IoT platforms.

- **Hardware development platforms** provide physical development boards for creating IoT devices, including microcontrollers, microprocessors, Systems on Chip (SoC), Systems on Module (SoM).
- **App development platforms** serve as an integrated development environment (IDE) with tools and features for coding applications.
- **Connectivity platforms** provide communication technologies to connect physical objects with the data center (on-premise or cloud) and transmit information between them. Among popular connectivity protocols and standards for the Internet of Things are MQTT, DDS, AMQP, Bluetooth, ZigBee, WiFi, Cellular, LoRaWAN and more.
- **Analytics platforms** use intelligent algorithms to analyze collected information and transform it into actionable insights for customers.

- **End-to-end IoT platforms** cover all aspects of IoT products, from development and connectivity to data management and visualization.

1.1.4 The Importance of IoT Cloud Services

Cloud computing is the predominant technology of our time that empowers numerous businesses and tech segments. The junction of Internet of Things and cloud services unleashes the potential of IoT devices to the fullest, opening new horizons for companies and customers.

Firstly, the cloud provides unlimited scalability, which is crucial as the demand for handling and storing Big Data from thousands of devices is continuously growing. Secondly, the cloud enables remote development and management, which is extremely convenient when connected assets are scattered across cities and countries.

Three types of cloud services are available for the development of the Internet of Things: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS).

As a rule, IoT cloud platforms are end-to-end solutions that combine capabilities such as app development, device management, connectivity management, data acquisition and storage, and data analysis and visualization.

1.1.5 Most Popular IoT Platforms in 2021

To make it easier for you to decide which IoT platform to choose for your project, we've compiled a list of the most popular Internet of Things platforms for this year, with detailed descriptions of each one.

1. Google Cloud IoT
2. Cisco IoT Cloud Connect
3. Salesforce IoT Cloud
4. IRI Voracity
5. Particle
6. IBM Watson IoT
7. ThingWorx
8. Amazon AWS IoT Core
9. Microsoft Azure IoT Hub
10. Oracle IoT

1. Google Cloud IoT



Google launched its platform for Internet of Things development on the basis of its end-to-end Google Cloud Platform. Currently, it's one of the world's top Internet of Things platforms. Google Cloud IoT is the integration of various services that add value to connected solutions.

- **Cloud IoT Core** allows you to capture and handle device data. A device manager component is used to register devices with the service, and monitor and configure them. MQTT and HTTP protocol bridges are used for device connection and communication with the Google Cloud Platform.
- **Cloud Pub/Sub** performs data ingestion and message routing for further data processing.
- **Google BigQuery** enables secure real-time data analytics.
- **AI Platform** applies machine learning features.
- **Google Data Studio** visualizes data by making reports and dashboards.
- **Google Maps Platform** helps visualize the location of connected assets.

The platform automatically integrates with Internet of Things hardware producers such as Intel and Microchip. It supports various operating systems, including Debian Linux OS.

Core features of Google Cloud IoT:

- AI and machine learning capabilities
- Real-time data analysis
- Strong data visualization
- Location tracking

Core use cases:

- Predictive maintenance
- Real-time asset tracking
- Logistics and supply chain management
- Smart cities and buildings

2. Cisco IoT Cloud Connect



Cisco IoT Cloud Connect is originally an offering for mobile operators. This mobility cloud-based software suite for industrial and individual use cases is on the list of the best Internet of Things cloud platforms. Cisco also provides reliable IoT hardware, including switches, access points, routers, gateways and more.

Take a look at some examples of powerful Cisco Internet of Things products and solutions.

- **Cisco IoT Control Center** ensures impeccable cellular connectivity management, allowing you to integrate all your IoT devices in one SaaS solution.
- **Extended Enterprise Solution** allows for the development of IoT business applications at the edge and ensures rapid deployment and centralized network management.
- **Edge Intelligence** simplifies data processing by allocating data flows either to local or multi-cloud environments.
- **Industrial Asset Vision** utilizes sensors to monitor your assets continuously and deliver data for better decision-making.
- **Cisco IoT Threat Defense** protects sensible data and devices against cyberattacks, providing secure remote access, segmentation, visibility and analysis, and other security services.

Core features of Cisco IoT Cloud Connect:

- Powerful industrial solutions
- High-level security
- Edge computing
- Centralized connectivity and data management
- Core use cases:

Connected cars

- Fleet management
- Home security and automation
- Payment and POS solutions
- Predictive maintenance
- Industrial networking
- Smart meters
- Healthcare

3. Salesforce IoT Cloud



Salesforce specializes in customer relations management and masterfully enhances this segment with the help of IoT solutions.

The Salesforce IoT Cloud platform gathers valuable information from connected devices to deliver personalized experiences to and build stronger relationships with your customers. It works in tandem with Salesforce CRM: data from connected assets is delivered directly to the CRM system where context-based actions are initiated immediately.

For example, if sensors detect an error in windmill performance, it is instantly reflected in the CRM dashboard and the system can either adjust parameters automatically or create a service ticket.

Core features of Salesforce IoT Cloud:

- Full integration of customers, products and CRM
- No need for programming skills to create rules, conditions and events due to a simple point-and-click UI
- Compatibility with third-party websites, services and other products
- A proactive approach to customer issues and needs

Core use cases:

- Government administration
- Machinery
- Financial services
- Marketing and advertising
- Chemicals

By using Salesforce IoT Cloud, businesses get a holistic view of customer data, improve customer experience and increase sales.

4. IRI Voracity



If you need an all-in-one data management platform that enables IoT data control at every stage of your business processes, IRI Voracity is the perfect fit.

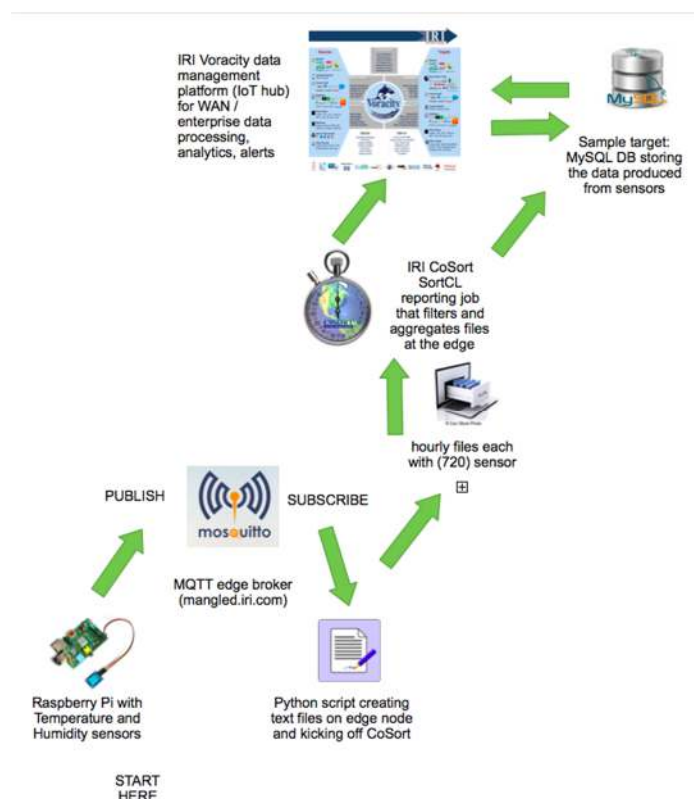
This platform uses two engines, IRI CoSort and Hadoop, to process Big Data. It can discover, govern, integrate, analyze, transform and migrate data from various sources and in various formats such as Unix, Linux or Windows file systems, ISAM, MongoDB, LDIF, HIVE, JSON, S3, PostgreSQL, MQTT, Kafka and more.

Core features of IRI Voracity:

- A **Data Governance Portal** enables data search and classification in silos. It also provides encryption and anonymization to comply with data privacy regulations.
- A **Faster ETL and Analytic Alternative** performs extraction and transformation of large-sized data much faster than legacy ETL tools.
- A **DB Ops Environment** allows you to administer all your databases from one place.

Core use cases:

- Big Data analytics
- ETL modernization
- Data governance



<https://www.iri.com/blog/business-intelligence/iri-iot-edge-aggregation/>

<https://www.youtube.com/watch?v=qkiIrnxo-48>

5. Particle



Particle offers an IoT edge-to-cloud platform for global connectivity and device management, as well as hardware solutions, including development kits, production modules and asset tracking devices. With Particle's team of IoT experts, who provide end-to-end professional services, you can develop your product from concept to production.

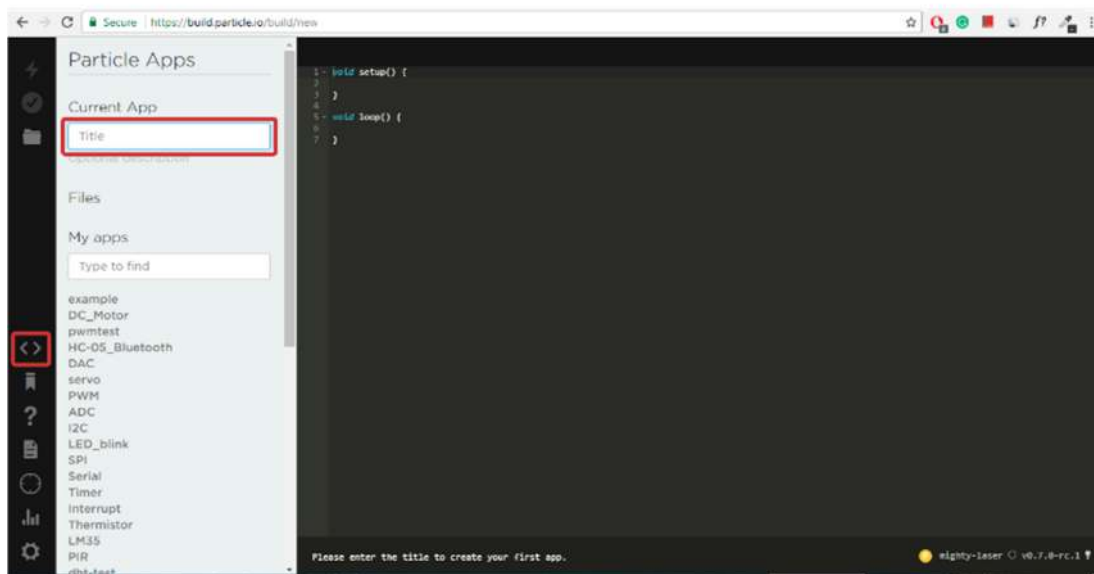
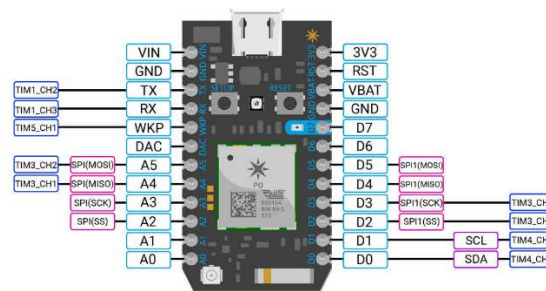
Core features of the Particle platform:

- Integration with third-party services via REST API
- Firewall-protected cloud
- Capability to work with data from Google Cloud or Microsoft Azure
- No need for technical expertise in order to use the platform

Core use cases:

- Real-time asset monitoring
- Live vehicle tracking
- Predictive maintenance
- Environmental monitoring
- Compliance monitoring
- Real-time order fulfillment

Particle Photon Board



- <https://www.electronicwings.com/particle/getting-started-with-particle-photon-with-ide>

6. IBM Watson IoT



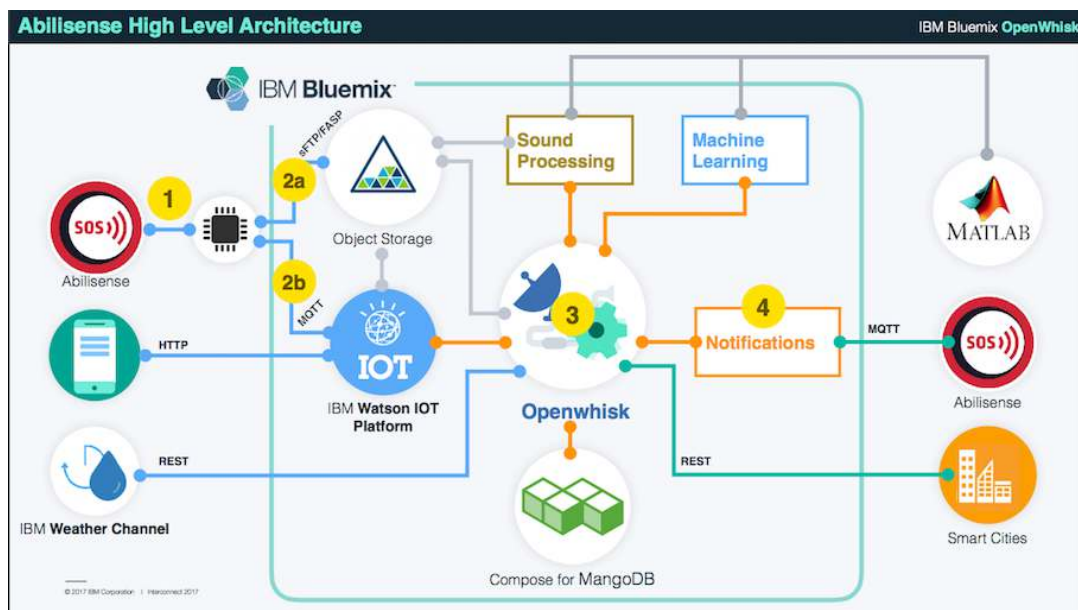
An IoT platform built on IBM Cloud is a fully managed cloud service for device management, flexible and scalable connectivity options, secure communications and data lifecycle management. With IBM Watson IoT, you can collect insights from automobiles, buildings, equipment, assets and things.

Core features of IBM Watson IoT:

- Data ingestion from any source with the help of MQTT
- Direct access to the latest data in the Cloudant NoSQL DB solution
- Built-in monitoring dashboards to control your assets
- Analytics Service to process raw metrics
- The Cloud Object Storage solution for long-term data archiving

Core use cases:

- Supply chain management
- Regulatory compliance
- Building management
- Energy consumption
- Shipping and logistics



- <https://www.linkedin.com/pulse/build-your-first-iot-application-ibm-watson-platform-janakiram-msv>
- <https://thenewstack.io/ibms-openwhisk-serverless/>

7. ThingWorx



The specialized Industrial Internet of Things (IIoT) platform ThingWorx is used in a variety of manufacturing, service and engineering scenarios. The platform addresses common challenges across industries, from remote monitoring and maintenance to workforce efficiency and asset optimization.

Core features of ThingWorx:

- Access to multiple data sources due to the extension of traditional industrial communications
- Powerful ready-to-use tools and applications to create and scale IIoT solutions quickly
- Real-time insights from complex industrial IoT data to proactively optimize operations and prevent issues
- Total control over network devices, processes and systems

Core Use Cases:

- Remote asset monitoring
- Remote maintenance/service
- Predictive maintenance and asset management
- Optimized equipment effectiveness



<https://metrosystems-des.com/thingworx/แนะนำ-thingworx-composer-และโครงสร้างใน-thingworx/>

8. Amazon AWS IoT Core



AWS IoT Core

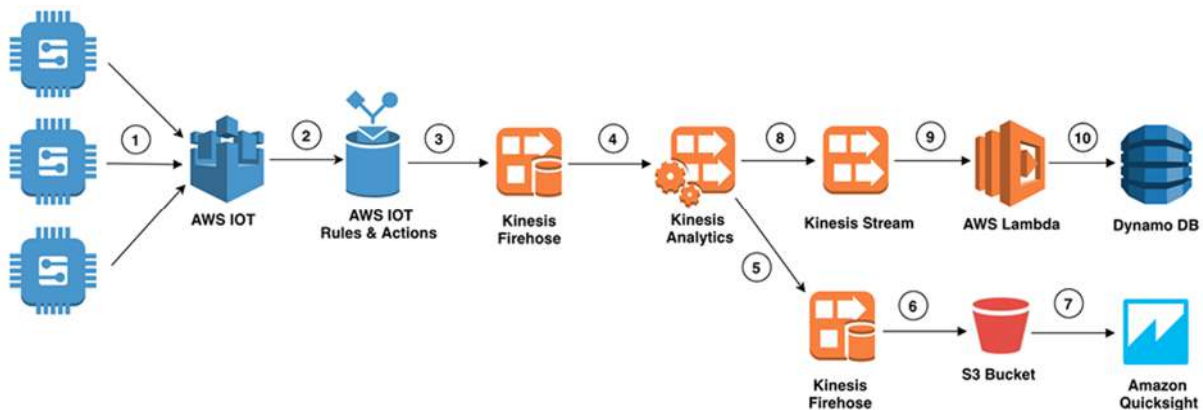
One of the leading players in the market, Amazon AWS IoT Core allows you to connect devices to AWS cloud services without the need to manage servers. The platform provides reliability and security for managing millions of devices.

Core features of Amazon AWS IoT Core:

- A wide choice of connection protocols, including MQTT, MQTT over WSS, HTTP and LoRaWAN
- Ability to use with other AWS services such as AWS Lambda, Amazon Kinesis, Amazon DynamoDB, Amazon CloudWatch, Alexa Voice Service and more to build IoT applications
- A high level of security provided by end-to-end encryption throughout all points of connection, automated configuration and authentication
- Machine learning capabilities
- A variety of services for edge computing

Core use cases:

- Connected vehicles
- Connected homes
- Asset tracking
- Smart building
- Industrial IoT



<https://blogs.itemis.com/en/a-serverless-iot-backend-with-aws-iot>

9. Microsoft Azure IoT Hub



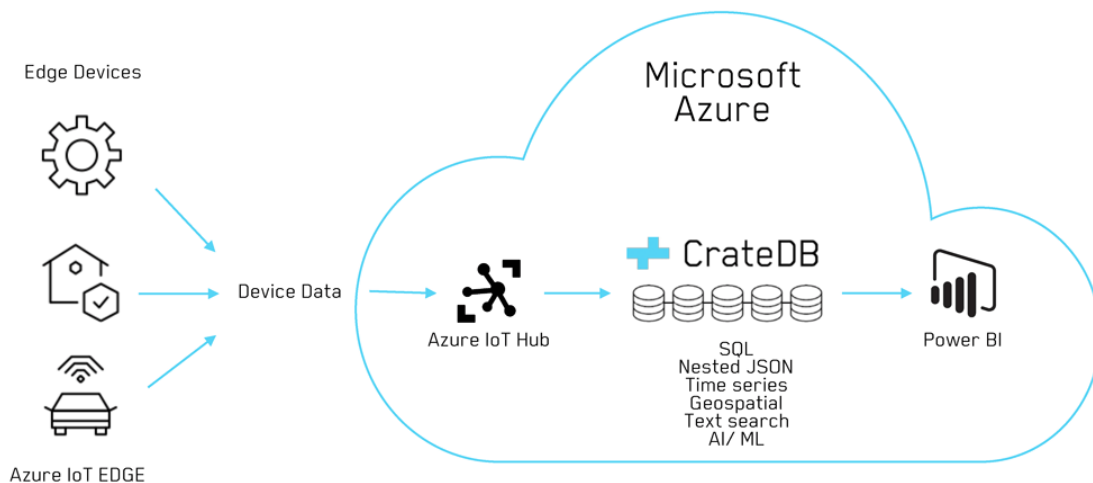
With the open-source Azure IoT platform from Microsoft, you can quickly build scalable and secure edge-to-cloud solutions. Utilizing ready-to-use tools, templates and services, you can develop flexible applications according to your company's needs.

Core features of Azure IoT Hub:

- Data protection all the way from the edge to the cloud
- The ability to operate even in offline mode with Azure IoT Edge
- Seamless integration with other Azure services
- Enhanced AI solutions
- Continuous cloud-scale analytics
- Fully managed databases
- Azure Industrial IoT solution

Core use cases:

- Automotive industry
- Discrete manufacturing
- Energy sector
- Healthcare
- Transportation
- Retail



10. Oracle IoT



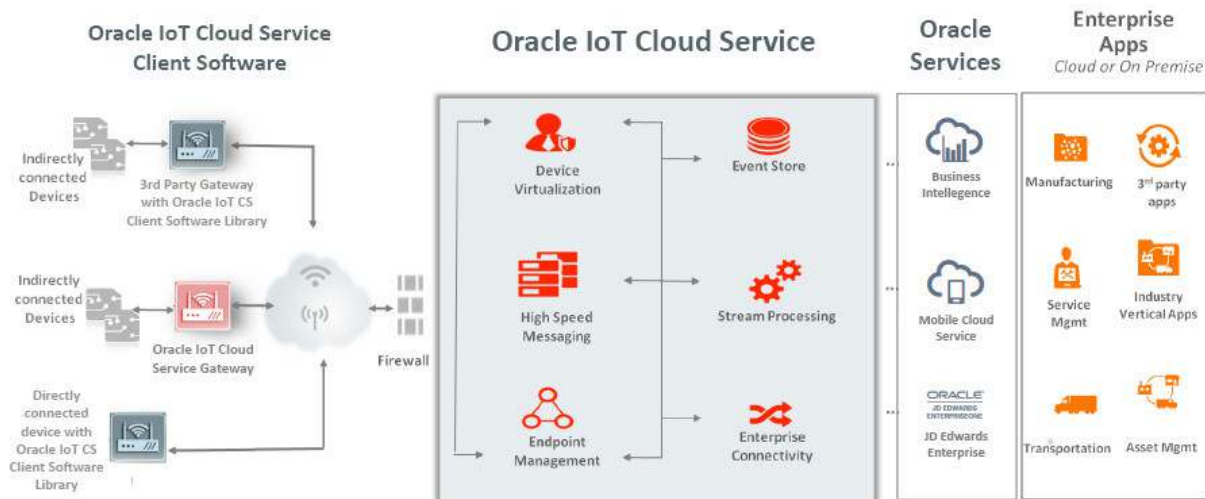
The Internet of Things Cloud Service by Oracle is a managed Platform as a Service (PaaS) for connecting your devices to the cloud.

Core features of Oracle IoT:

- The ability to create applications and connect them to devices with JavaScript, Java, Android, iOS, C POSIX and REST APIs
- Integration with enterprise applications, web services and other Oracle Cloud Services
- Real-time analysis tools to aggregate and filter incoming data streams
- Automatic synchronization of data streams with Oracle Business Intelligence Cloud Service
- Unique digital identity for each device to establish trust relationships among devices and applications

Core use cases:

- Connected logistics
- Predictive maintenance
- Smart manufacturing
- Workplace safety



1.1.6 How to Choose the Best IoT Platform

There's no definite answer to this question since there's no one best platform suitable for any digital project. The choice will always depend on the specific requirements of your business.

Large enterprises are more likely to turn to giants such as Amazon or Microsoft. Their offerings are the best established, but also the most expensive. Smaller companies may find more cost-efficient options that will nevertheless perfectly meet their requirements.

When choosing a provider, you should consider the technical capabilities of a platform, its partner ecosystem, industry-specific features and, in general, the provider's reputation. All these parameters should comply with your company strategy and budget.

If you need help selecting a platform, contact SaM Solutions' specialists. Our development teams have experience in building IoT applications and devices and can advise you on all related issues. We know the pros and cons of various platforms, and will easily be able to recommend the right option for your digital strategy.

1.1.7 Frequently Asked Questions (FAQ)

1. What Are the Top IoT App Development Platforms?

Currently, there are more than 600 publicly known Internet of Things platforms globally. However, the leaders — *Amazon AWS IoT Core, Microsoft Azure IoT Hub and IBM Watson* — still hold their positions as the top three Internet of Things app development platforms.

2. Why do I need an IoT platform?

An IoT platform is a unique tool that will provide *continuous monitoring of all your assets, be it vehicles, manufacturing equipment, livestock*, or anything else. It will help you as the owner of a business gain a comprehensive view of all processes seasoned with intelligent analytics of collected data. The result — quicker decisions, reduced issues and increased revenues.

3. What Is the Difference between IoT and Cloud Computing?

IoT is about gathering data from physical devices and transferring it to digital space for further analysis. Cloud computing is purely about data processing, delivery and storage. These are two different technologies that complement each other, resulting in efficient solutions.

1.2 Top 10 IoT Platforms for 2021 – Start up project

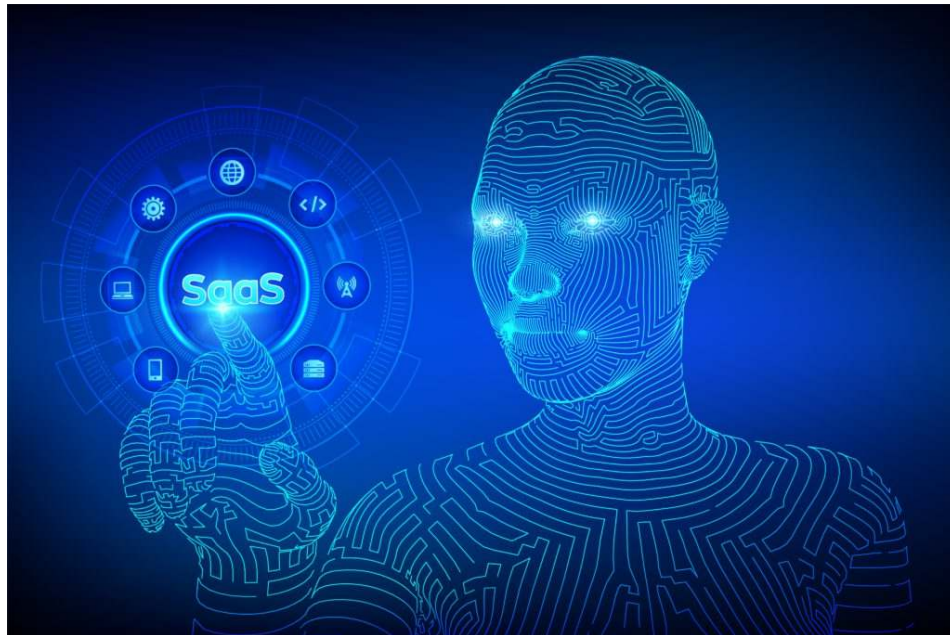
<https://ifra.io/iot-platform-top-10-ในปี-2021/>

ธันวาคม 8, 2020 [SHINJI IoT Platform](#)



ในช่วงไม่กี่ปีที่ผ่านมาเราได้เห็น IoT และศักยภาพที่ไร้ขอบเขตของมัน แม้แต่มือถือที่เราใช้อยู่ถือเป็นอุปกรณ์ IoT ที่ทำให้เราได้เชื่อมต่อกับโลกอินเทอร์เน็ตแบบไร้พรมแดน

ด้วยการเปลี่ยนแปลงอย่างรวดเร็วในปี 2020 ในทุกๆ อุตสาหกรรมบนโลกนี้ นี่ทั้งในโรคของ Covid19 ที่มาเร่งความเร็วของเรื่องเทคโนโลยีขึ้นไปอีก ทำให้ภาคส่วนของ IoT นั้นเติบโตขึ้นอย่างมหาศาล



กลับมาดูพื้นฐานทางด้าน IoT หากเราอยากจะสร้าง สิ่งๆ หนึ่งในด้าน IoT เราต้องมีพื้นฐานหลายๆ อย่าง ทั้งด้านเครือข่าย Server, การส่งข้อมูล [MQTT](#) และอีกมากมาย

แต่ปัจจุบันเรามีตัวช่วยในการทำโปรเจก IoT หรือ สิ่งๆ หนึ่งขึ้นมาอย่างรวดเร็วและง่ายภายในเวลาไม่ถึง 1 วัน เพื่อสร้างโปรเจก IoT ของเราเอง เช่น ระบบรดน้ำต้นไม้อัตโนมัติ ในหน้าบ้าน สิ่งเหล่านี้ หากมี Platform IoT ที่ช่วยเชื่อมต่อ Board หรืออุปกรณ์ของเราจะทำให้เราสร้างโปรเจกของเราได้อย่างรวดเร็วมากๆ

Platform IoT นั้นมีหลายหลายมากๆ ในตลาดทั้งใช้สำหรับสร้างระบบใหญ่ๆ ที่ใช้ในองค์กร เช่น Google Cloud, Amazon แต่ Platform พวกนี้มักจะใช้งานยากและต้องเป็นผู้เชี่ยวชาญเฉพาะด้าน อีกทั้งยังใช้เวลาศึกษานานมากๆ ด้วย



สำหรับ IoT Platform ในปี 2021 ที่เราเลือกมานี้ จะสามารถช่วยให้เราสามารถทำโปรเจกได้อย่างรวดเร็วและง่ายดาย จากไอเดียเราภายในเวลาสั้นๆ มาดูกันว่ามืออะไรบ้าง โดยการจัดอันดับครั้งนี้เราจะเลือกจากเงื่อนไขดังต่อไปนี้

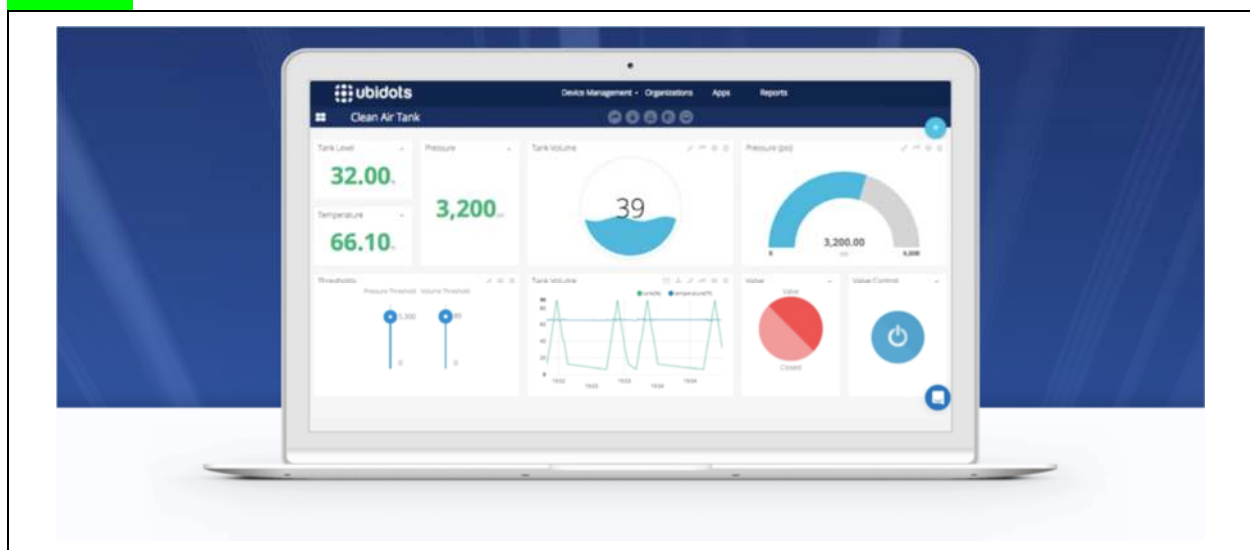
- ใช้งานง่าย สำหรับคนไม่มีพื้นฐานทางด้าน IoT สามารถใช้งานได้
- สามารถสร้างโปรเจกด้าน IoT ได้หลากหลายและรวดเร็ว
- มีคู่มือการใช้งานสามารถอ่านและทำตามได้ง่าย
- ความคุ้มค่าและราคา โดยเทียบแต่ละแพ็คเกจอันไหนราคาถูกและคุ้มค่าที่สุด

1. Thingsboard



Thingsboard เป็น Open-source สามารถใช้งานได้ฟรีด้วย Start ใน [Github](https://github.com) ถึง 7,700 ดาว สามารถเชื่อมต่อได้หลากหลายและใช้งานง่าย

2. Ubidots



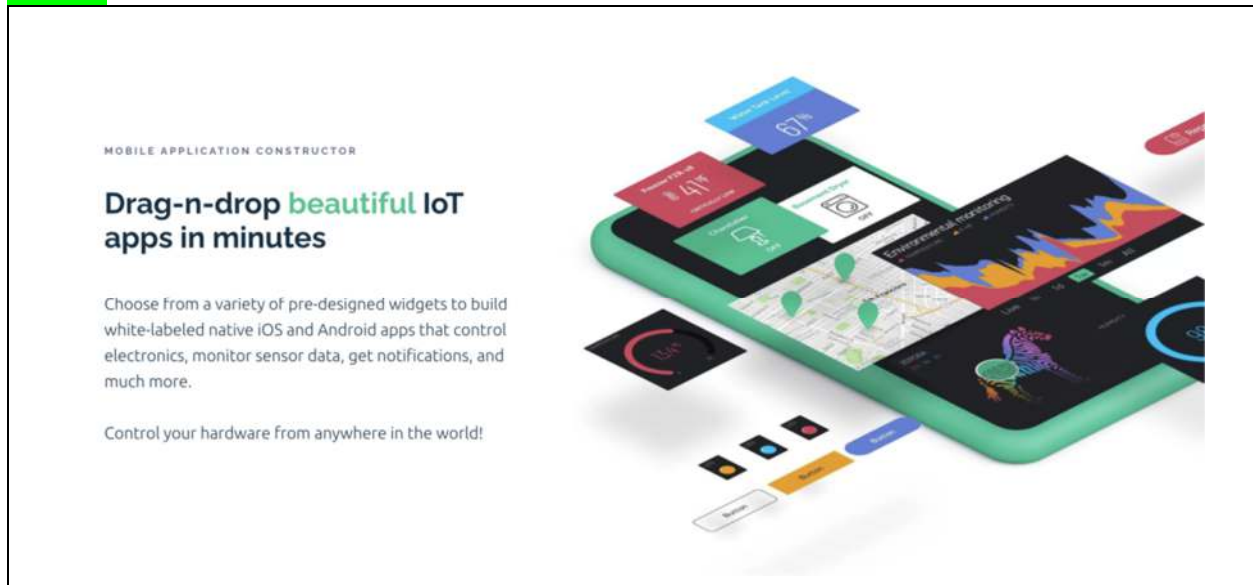
เป็นเว็บแอปพลิเคชัน สามารถสร้าง Internet of Things (IoT) อย่างรวดเร็วโดยไม่ต้องเขียนโค้ดหรือจ้างทีมพัฒนาซอฟต์แวร์ ส่วนราคาเริ่มต้นนั้นแพงกว่าเจ้าอื่นพอสมควรเริ่มต้นที่ \$49 แต่โดยรวม UI ใช้งานได้ง่ายมาก

3. Thinger



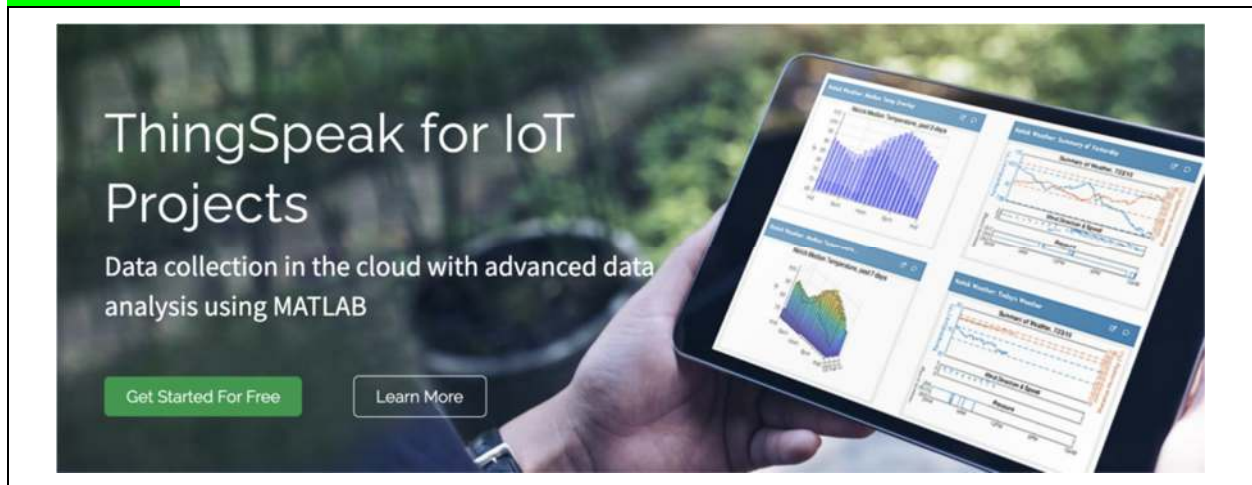
Thinger สามารถสร้างต้นแบบปรับขนาดและจัดการผลิตภัณฑ์ IoT จุดเด่นคือเป็น Open Source สามารถใช้งานได้ฟรี

4. Blynk



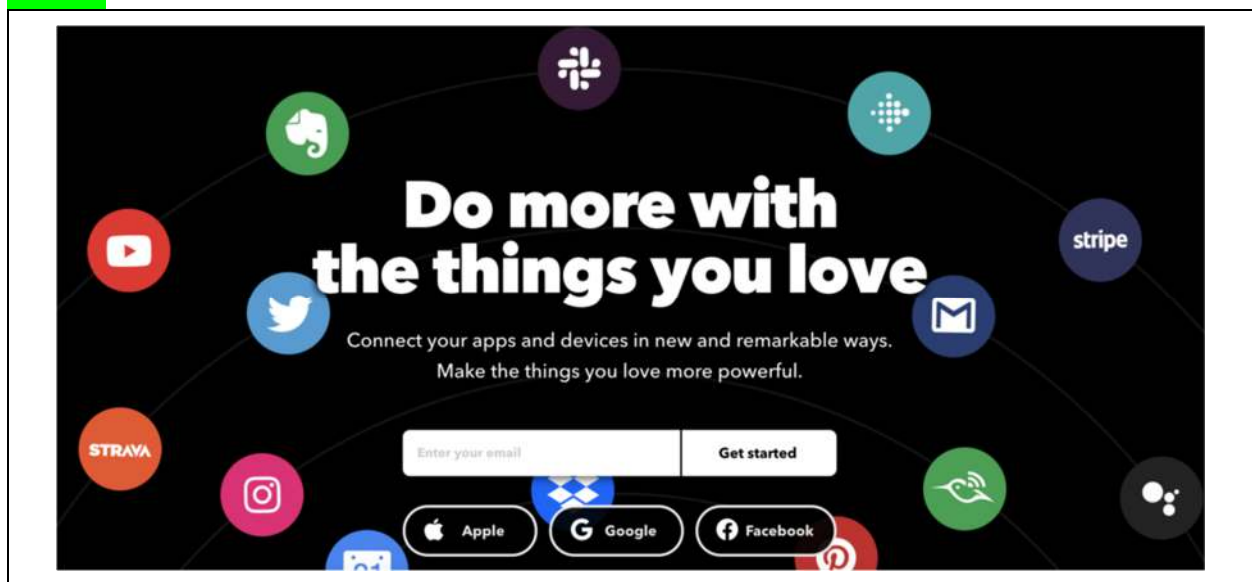
BlynK นั้นใช้งานได้ง่าย มากกว่า Platform ทั้งหมด โดยข้อดีคือคลิกลากวาง ไม่กี่ Step เหมาะสำหรับทำ Prototype เบื้องต้น ข้อเสียคือใช้ได้แค่ใน Mobile ไม่สามารถใช้งานได้บน Browser

5. Thingspeak



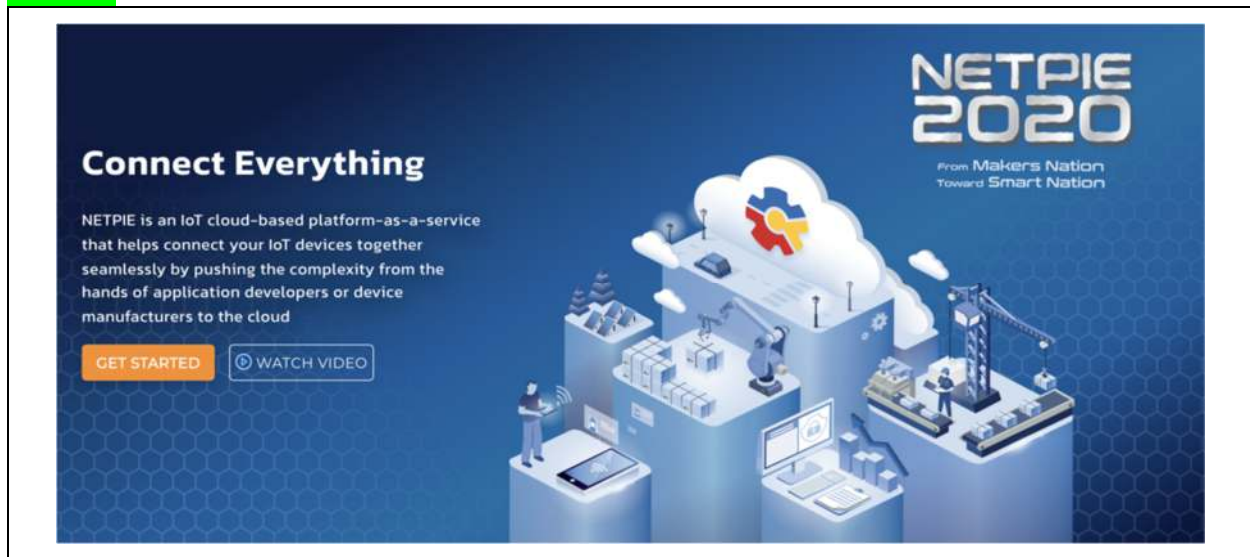
Thingspeak เป็น platform รวมข้อมูลในระบบคลาวด์ด้วยการวิเคราะห์ข้อมูลขั้นสูงโดยใช้ MATLAB โดยด้านการวิเคราะห์ข้อมูลแบบละเอียดและแม่นยำ จะอยู่เหนือ Platform อื่นๆทั้งหมด

6. IFTTT



IFTTT สามารถเชื่อมต่ออุปกรณ์ IoT ไปยัง อุปกรณ์ IoT อีกเครื่องหนึ่งได้อย่างง่ายดายเช่นเชื่อมต่อมือถือหรือแจ้งเตือนเปิด APP Devices ต่าง ๆ ได้อย่างง่ายดาย

7. Netpie



Netpie เป็น Platform ไทย ที่สนับสนุนโดย nectec โดย Netpie สามารถใช้งานได้ฟรี ะกับหนึ่ง สามารถเชื่อมต่ออุปกรณ์ได้ง่าย มี Community หลายหมื่นคน ที่เข้มแข็ง ที่ช่วยกันตอบปัญหาและคำถามต่างๆ

8. Kaa



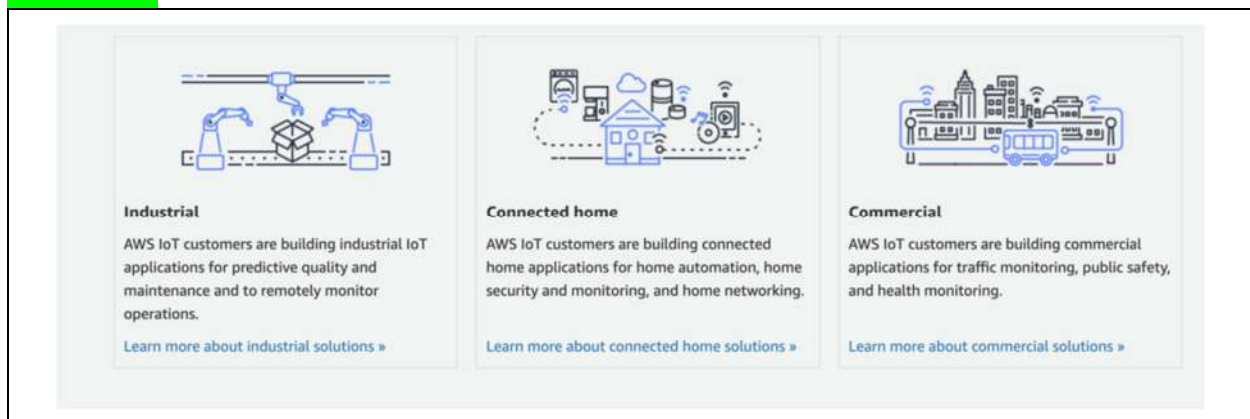
Kaa project มี solution platform ที่เยอะมากๆ สามารถตอบโจทย์การสร้างโปรเจค IoT ที่หลากหลาย ข้อเสียคือใช้งานยากและมีราคาแพงเหมาะสำหรับองค์กร หรือบริษัท ใหญ่ๆ ที่ต้องการตอบโจทย์ solution

9. Particle



Particle ถึงแม้จะใช้งานได้ ง่ายๆ หน่อย แต่ราคาไม่แพง เริ่มต้นที่ \$2.99 และทดลองใช้งานได้ฟรี 3 เดือนเลยทีเดียว ข้อเสียคือต้องเชื่อมต่อ Hardware ของ Particle เอง

10. AWS IoT



หากต้องการทำ IoT Project ขนาดใหญ่ ขอแนะนำ AWS IoT ที่สามารถตอบโจทย์ได้หลากหลายทุกอุตสาหกรรม และมีฟังก์ชัน Support เยอะมากๆ ข้อเสียคือ ราคาค่อนข้างแพง และใช้งานยากสำหรับผู้ที่ไม่มีความรู้พื้นฐาน IoT และการเขียนโปรแกรม

2/5 -- ThingsBoard Rule Chains and Alarm

Lab401 -- Getting Started with Rule Engine

1. What is ThingsBoard Rule Engine?

Rule Engine is an easy-to-use framework for building event-based workflows. There are 3 main components:

- **Message** - any incoming event. It can be an incoming data from devices, device life-cycle event, REST API event, RPC request, etc.
- **Rule Node** - a function that is executed on an incoming message. There are many different Node types that can filter, transform or execute some action on incoming Message.
- **Rule Chain** - nodes are connected with each other with relations, so the outbound message from rule node is sent to next connected rule nodes.

2. Typical Use Cases

ThingsBoard Rule Engine is a highly customizable framework for complex event processing. Here are some common use cases that one can configure via ThingsBoard Rule Chains:

- Data validation and modification for incoming telemetry or attributes before saving to the database.
- Copy telemetry or attributes from devices to related assets so you can aggregate telemetry. For example, data from multiple devices can be aggregated in related Asset.
- Create/Update/Clear alarms based on defined conditions.
- Trigger actions based on device life-cycle events. For example, create alerts if Device is Online/Offline.
- Load additional data required for processing. For example, load temperature threshold value for a device that is defined in Device's Customer or Tenant attribute.
- Trigger REST API calls to external systems.
- Send emails when complex event occurs and use attributes of other entities inside Email Template.
- Take into account User preferences during event processing.
- Make RPC calls based on defined condition.
- Integrate with external pipelines like Kafka, Spark, AWS services, etc.

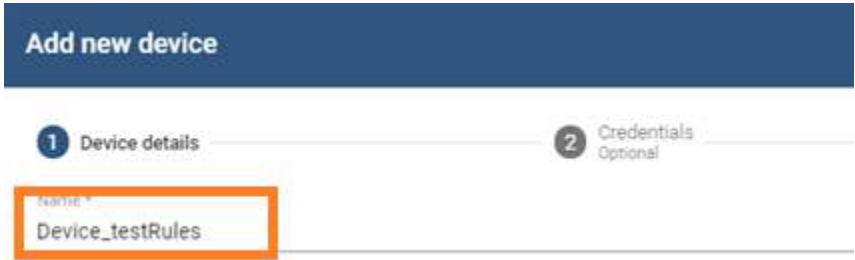
3. Hello-World Example

Let's assume your device is using DHT22 sensor to collect and push temperature to the ThingsBoard. DHT22 sensor can measure temperature from -40°C to +80°C.

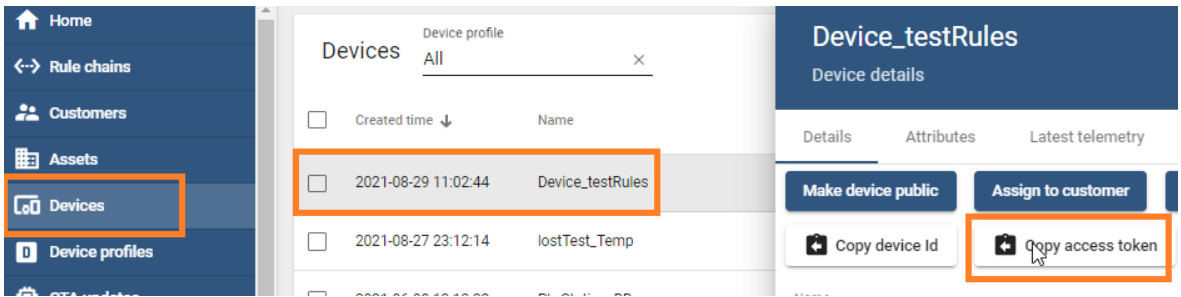
In this tutorial we will configure ThingsBoard Rule Engine to store all temperature within -40 to 80°C range and log all other readings to the system log.

3.1 Create Device

Add new Device



Get Token ID



MdAx9wW7GoLeishzcw0

3.2 Test with CURL

Curl Command test is

```
curl -v -X POST -d '{"temperature":99}'
http://demo.thingsboard.io:8080/api/v1/$ACCESS_TOKEN/telemetry --header "Content-Type:application/json"
```

Test curl on line from <https://reqbin.com/curl>

Run Curl Commands Online

Execute Curl commands directly from your browser. Learn Curl with live Curl examples. Test APIs with

File Generate Code Tools Share

Curl Raw US Run

1. Command

```
curl -v -X POST -d '{"temperature":99}'
http://demo.thingsboard.io/api/v1/MdAxA9wW7GoLeishzcw0/telemetry --header "Content-Type:application/json"
```

2. Run

3. Return = 200 is success
Status: 200 ()

Content Header

```
curl -v -X POST -d '{"temperature":99}'
http://demo.thingsboard.io/api/v1/MdAxA9wW7GoLeishzcw0/telemetry --header "Content-Type:application/json"
```

Data in Device – Latest telemetry

Device_testRules

Device details

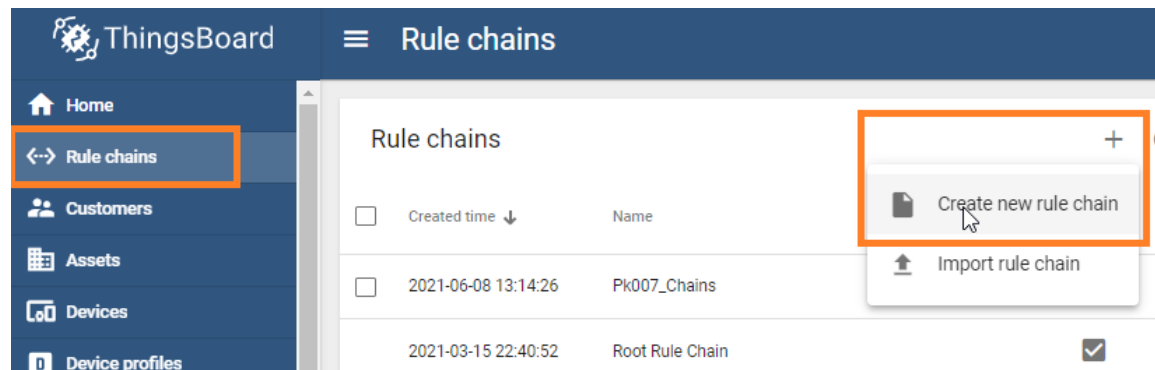
Details Attributes Latest telemetry Alarms Events Relations

Latest telemetry

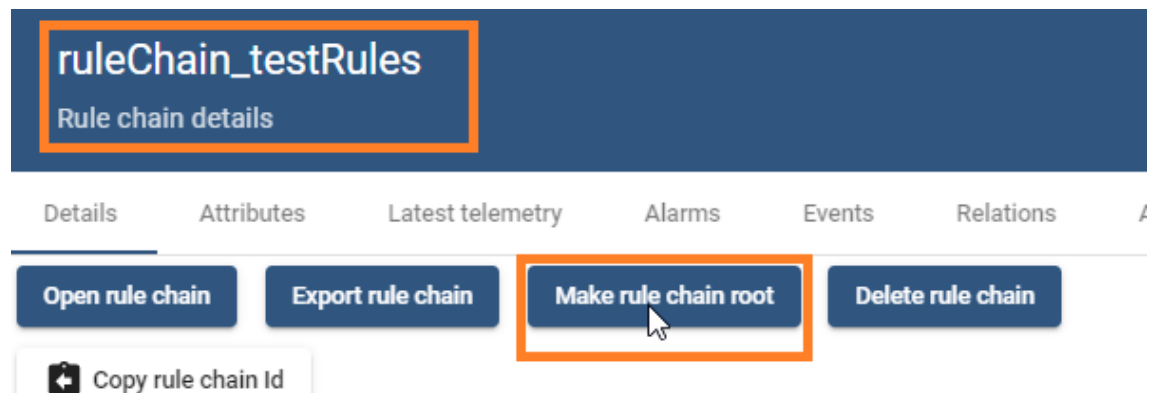
Last update time	Key ↑	Value
2021-08-29 11:09:17	temperature	99

3.3 Add Rule Engine and Filtering Input Data

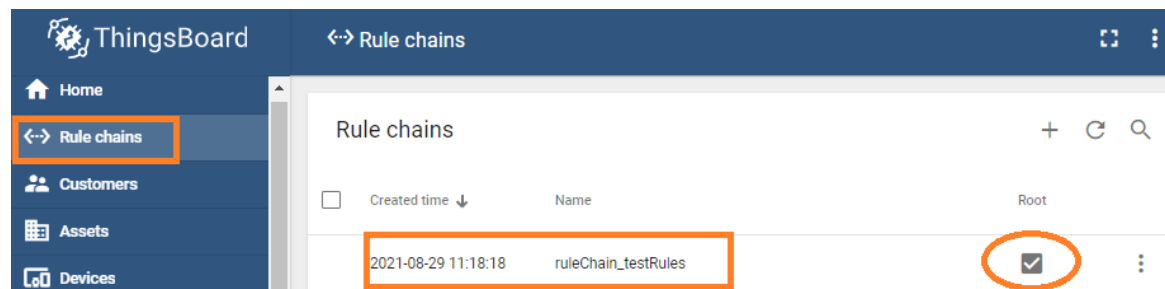
Create new rule chain, Name is **ruleChain_testRules**



Setting to root



Root rule chain



Edit and configure rule



- Node Input
- Node Message type switch name: msgSw_testRules
- Node Scrip name: scrp_testRules
- Node Save timeseries name: saveTS_testRules
- Chain Post telemetry
- Chain True

Scrip

scrip_testRules

Filter

```
function Filter(msg, metadata, msgType) {
```

```
1 return typeof msg.temperature === 'undefined'
2    || (msg.temperature >= -40 && msg.temperature <= 80);
3 }
```

```
return typeof msg.temperature === 'undefined'
    || (msg.temperature >= -40 && msg.temperature <= 80);
```

Test Script Function

Test script function

Message type *
Post telemetry

Message

```
{
  "temperature": 99,
  "humidity": 78
}
```

test temperature

Filter

```
function Filter(msg, metadata, msgType) {
  1 return typeof msg.temperature === 'undefined'
  2    || (msg.temperature >= -40 && msg.temperature <= 80);
  3 }
}
```

Output

```
1 false
```

3. result

Test 2.Test

Cancel Save

3.4 Test with CURL again

Send 55

```
curl -v -X POST -d '{"temperature":55}'
http://demo.thingsboard.io/api/v1/MdAxA9wW7GoLeishzcw0/tel
emetry --header "Content-Type:application/json"
```

Get 55

Last update time	Key ↑	Value
2021-08-29 11:28:51	temperature	55

Send 99

```
curl -v -X POST -d '{"temperature":99}'
http://demo.thingsboard.io/api/v1/MdAxA9wW7GoLeishzcw0/tel
emetry --header "Content-Type:application/json"
```

Last data is 55

Last update time	Key ↑	Value
2021-08-29 11:28:51	temperature	55

Send 22

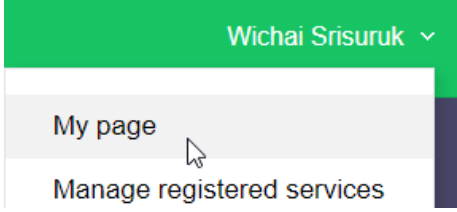
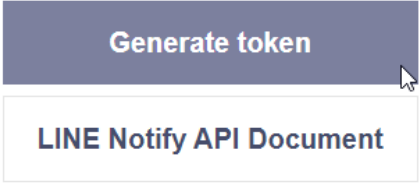
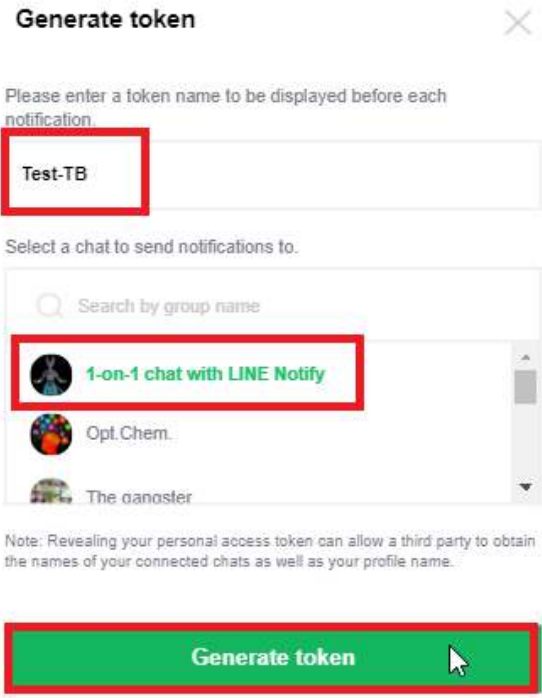
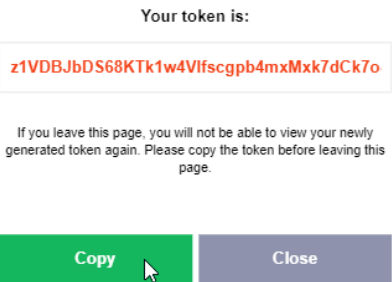
```
curl -v -X POST -d '{"temperature":22}'
http://demo.thingsboard.io/api/v1/MdAxA9wW7GoLeishzcw0/tel
emetry --header "Content-Type:application/json"
```

Last data is 22

Last update time	Key ↑	Value
2021-08-29 12:19:27	temperature	22

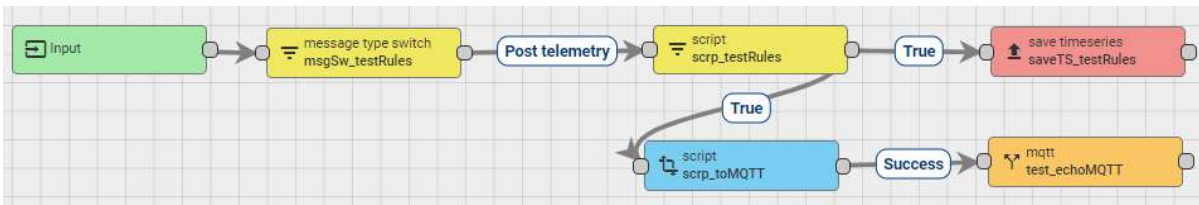
4. If xxx Then LINE Notify Alert Example

4.1 Login LINE -> <https://notify-bot.line.me/en/> and get Token Key

	Select → My page
	Click → Generate TOKEN
	Fill Name 1-on-1 for Test Generate Token
	Copy

4.2 setting rule chain for forward data to external broker

Add (1) scrip, (2) mqtt



(1) Set Scrip → name = scrp_toMQTT

Name *

scrp_toMQTT

Transform

function Transform(msg, metadata, msgType) {

```

# 1 var newMsg = []
# 2     newMsg = "Temp=" + msg.temperature + ", Hudmid=" + msg.humidity
3
4 return {msg: newMsg, metadata: metadata, msgType: msgType};

```

var newMsg = []

newMsg = "Temp=" + msg.temperature + ", Hudmid=" + msg.humidity

return {msg: newMsg, metadata: metadata, msgType: msgType};

(2) Set mqtt → name = test_echoMQTT, Topic = monitorTB, Host = test.mosquitto.org:1883

Name *

test_echoMQTT

Topic pattern *

monitorTB

Hint: use \${metadataKey} for value from metadata, \${messageKey} for value from messa

Host *

test.mosquitto.org

Port *

1883

Test with MQTTLens → Host Name, Topic Setting

Hostname

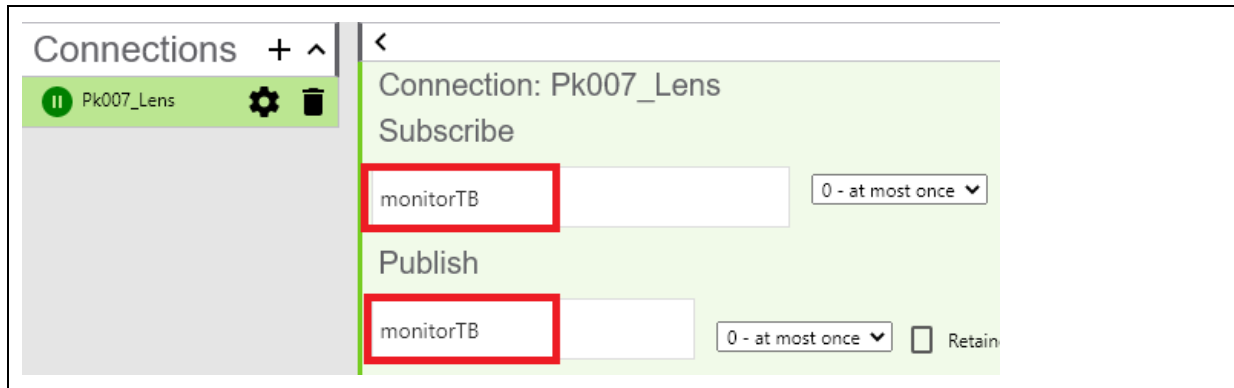
tcp://



test.mosquitto.org

Port

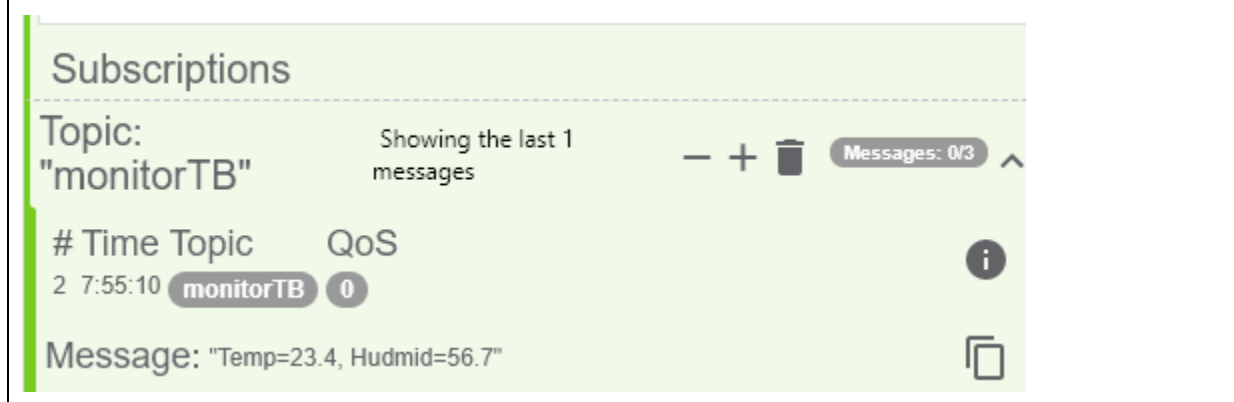
1883



4.3 Test forward MQTT with CURL Command

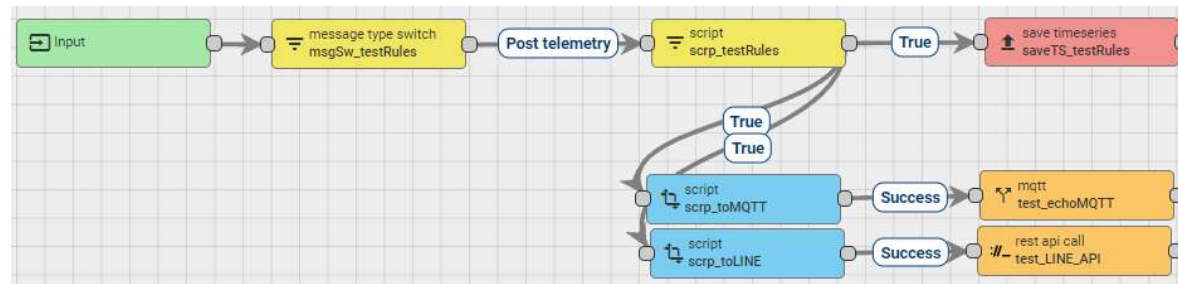
```
curl -v -X POST -d '{"temperature":23.4,"humidity": 56.7}'
http://demo.thingsboard.io/api/v1/MdAxA9wW7GoLeishzcw0/telemetry --header "Content-Type:application/json"
```

```
curl -v -X POST -d '{"temperature":23.4,"humidity": 56.7}'
http://demo.thingsboard.io/api/v1/MdAxA9wW7GoLeishzcw0/telemetry --header "Content-Type:application/json"
```



4.4 setting rule chain for LINE Alert

Add (1) scrip, (2) rest api call



(1) Set Scrip → name = scrp_toLINE

Name *

scrp_toLINE

Transform

```
function Transform(msg, metadata, msgType) {
```

```

1  var newMsg = "Overheat, Temperature = " + msg.temperature + "'C";
2  var newmetadata = { message: newMsg };
3  var msgType = "Debug Mode";
4  return {msg: newMsg, metadata: newmetadata, msgType: msgType};

```

```
var newMsg = "Overheat, Temperature = " + msg.temperature + "'C";
```

```
var newmetadata = { message: newMsg };
```

```
var msgType = "Debug Mode";
```

```
return {msg: newMsg, metadata: newmetadata, msgType: msgType};
```

(2) Set rest api call → name test_LINE_API

Name *
test_LINE_API

Endpoint URL pattern *
https://notify-api.line.me/api/notify?message=\${message}

Hint: use \${metadataKey} for value from metadata, \${messageKey} for value from message body

Request method
POST

☐ Enable proxy

☐ Use simple client HTTP factory

Header

Value

Content-Type

application/x-www-form-urlencoded

Authorization

Bearer xziSiD2pEFshJr8699kikWnV2R3THft9jYV

https://notify-api.line.me/api/notify?message=\${message}

POST

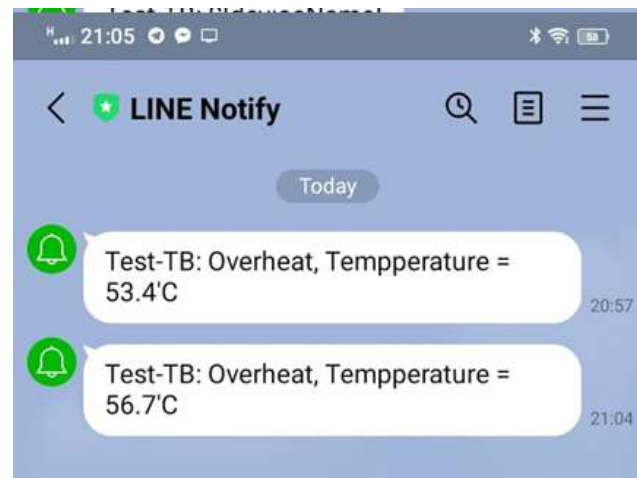
Content-Type application/x-www-form-urlencoded

Authorization Bearer LINE-API-Key-zzzzzzzzzz

4.5 CURL Test for LINE alert

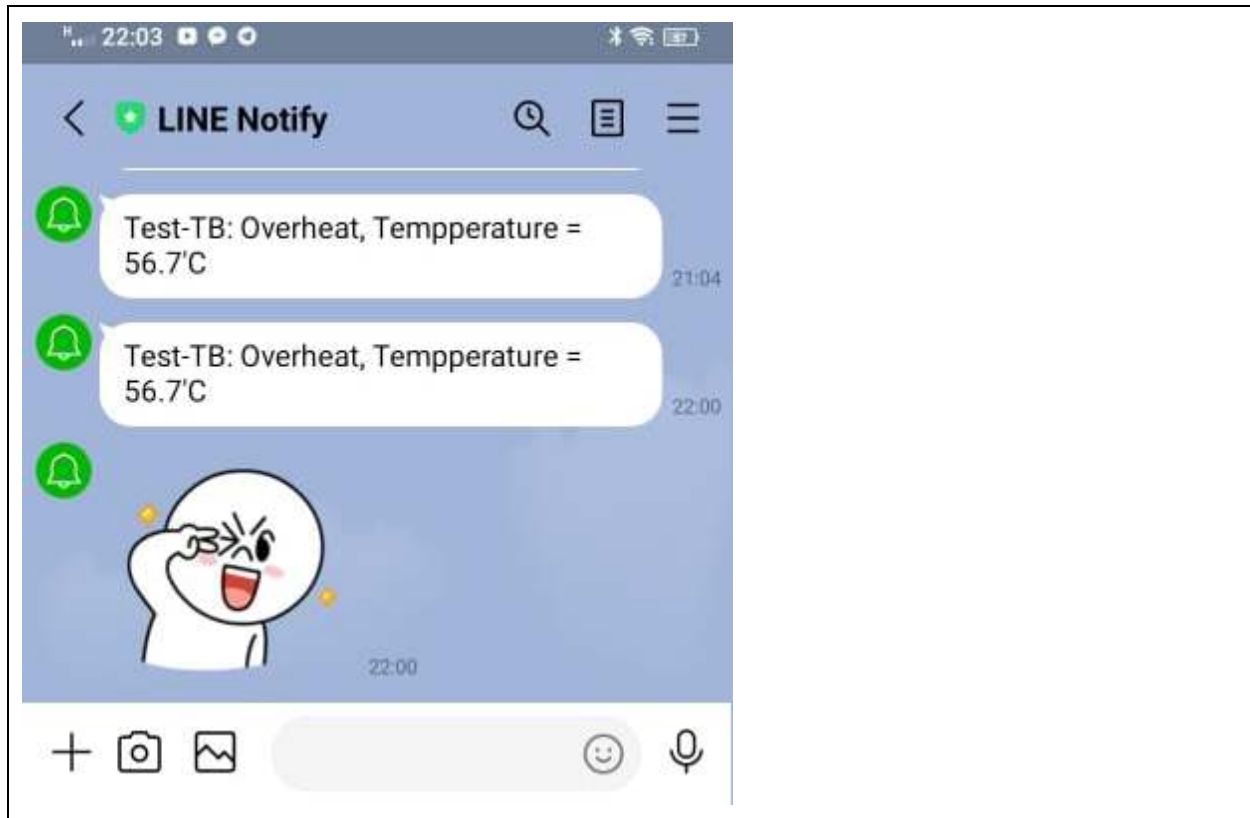
```
curl -v -X POST -d '{"temperature":56.7,"humidity": 76.5}'  
http://demo.thingsboard.io/api/v1/MdAxA9wW7GoLeishzcw0/telemetry --header "Content-Type:application/json"
```

```
curl -v -X POST -d '{"temperature":56.7,"humidity": 76.5}'  
http://demo.thingsboard.io/api/v1/MdAxA9wW7GoLeishzcw0/telemetry --header "Content-Type:application/json"
```

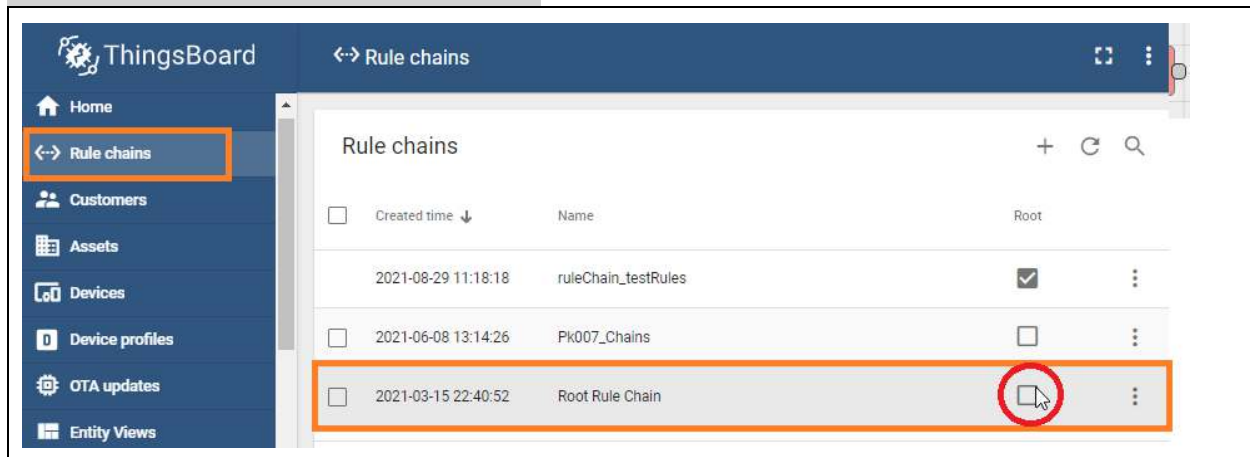


4.5 LINE with Sticker

Name * scrp_toLINE	
Transform <pre>function Transform(msg, metadata, msgType) { 1 var newMsg = "Overheat, Tempperature = " + msg.temperature + "'C"; 2 var newmetadata = { 3 LN_message: newMsg, 4 LN_stickerPack : 1, 5 LN_stickerID : 106 6 }; 7 var msgType = "Debug Mode"; 8 return {msg: newMsg, metadata: newmetadata, msgType: msgType}; }</pre>	
<pre>var newMsg = "Overheat, Tempperature = " + msg.temperature + "'C"; var newmetadata = { LN_message: newMsg, LN_stickerPack : 1, LN_stickerID : 106 }; var msgType = "Debug Mode"; return {msg: newMsg, metadata: newmetadata, msgType: msgType};</pre>	
Name * test_LINE_API	<input type="checkbox"/> Debug mode
Endpoint URL pattern * https://notify-api.line.me/api/notify?message=\${LN_message}&stickerPackageId=\${LN_stickerPack}&stick	
Hint: use \${metadataKey} for value from metadata, \${messageKey} for value from message body	
https://notify-api.line.me/api/notify?message=\${LN_message}&stickerPackageId=\${LN_stickerPack}&stickerId=\${LN_stickerID}	



5 End of testing set default rule chain root



Lab402 – Create and Clear Alarm

1 Use case

Let's assume your device is using DHT22 sensor to collect and push temperature readings to ThingsBoard. DHT22 sensor is good for -40 to 80°C temperature readings. We want generate Alarms if temperature is out of good range.

In this tutorial we will configure ThingsBoard Rule Engine to

- Create or Update existing Alarm if temperature > 80°C or temperature < -40°C
- Clear Alarm if temperature > -40°C and < 80°C

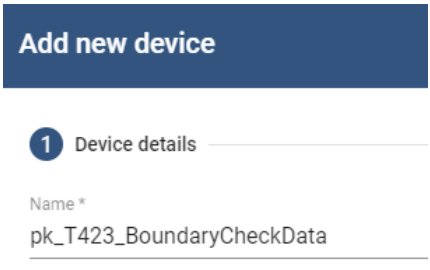
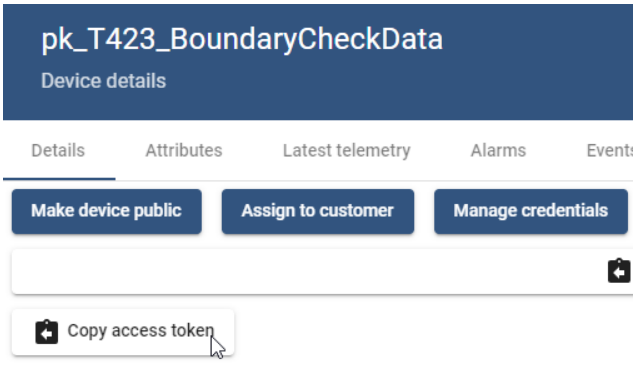
2. Prerequisites

We assume you have completed the following guides and reviewed the articles listed below:

- [Getting Started](#) guide.
- [Rule Engine Overview](#).





3. Adding the device

Add Device entity in ThingsBoard.

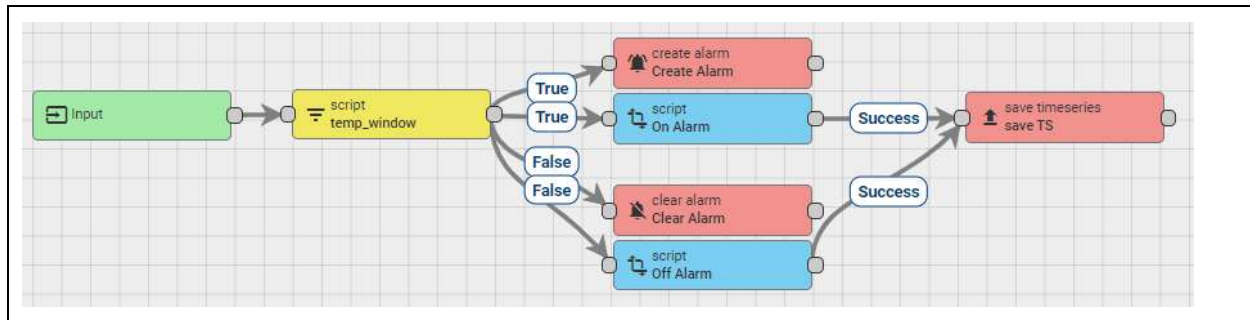
	<p>Name = pk_T423_BoundaryCheckData</p>
	<p>Copy Access Token A3MMKatweoQ4IpdsMFr2</p>

4. Adding the Rule Chain and Configure

4.1 Create “pkT423_Set&Clear Alarm”

 <p>Name *</p> <p>pkT423_Set&Clear Alarm</p> <p><input type="checkbox"/> Debug mode</p>	<p>Add New Rule Chain</p> <p>Name = pkT423_Set&Clear Alarm</p>
<p><input type="checkbox"/> 2021-08-29 22:44:39 Create & Clear Alarms <input type="checkbox"/>   </p>	<p>Edit → Create & Clear Alarms</p>
<p>Name *</p> <p>boundWindows</p> <p><input type="checkbox"/> Debug mode</p> <p>Filter</p> <p>function Filter(msg, metadata, msgType) { Tidy</p> <pre>1 return msg.temperature < -40 msg.temperature > 80;</pre> <p>return msg.temperature < -40 msg.temperature > 80;</p>	<p>Add (1/6)</p> <p>Filter Scrip</p> <p>Name = boundWindows</p>
<p>Name *</p> <p>Create Alarm <input type="checkbox"/> Debug mode</p> <p>Alarm details builder Tidy</p> <p>function Details(msg, metadata, msgType) {</p> <pre>1 var details = {}; 2 if (metadata.prevAlarmDetails) { 3 details = JSON.parse(metadata.prevAlarmDetails); 4 } 5 return details;</pre> <p>Test details function</p> <p><input type="checkbox"/> Use message alarm data <input type="checkbox"/> Use dynamically change th</p> <p>Alarm type *</p> <p>Critical Temperature</p> <p>Alarm severity</p> <p>Critical</p> <p>Hint: use \${metadatakey} for value from metadata, \${messagekey} for value from message body</p> <p><input checked="" type="checkbox"/> Propagate</p>	<p>Add (2/6)</p> <p>Create Alarm</p> <p>Name = Create Alarm</p> <p>{default}</p> <p>Critical Temperature</p>

<p>Name * <input type="checkbox"/> Debug mode</p> <p>Clear Alarm</p> <hr/> <p>Alarm details builder</p> <p>function Details(msg, metadata, msgType) {</p> <pre> 1 var details = {}; 2 if (metadata.prevAlarmDetails) { 3 details = JSON.parse(metadata.prevAlarmDetails); 4 } 5 return details; </pre> <p>Tidy</p> <p>Test details function</p> <p>Alarm type *</p> <p>Critical Temperature</p> <p>Hint: use \${metadataKey} for value from metadata, \${messageKey} for value from message</p>	<p>Add (3/6)</p> <p>Clear Alarm</p> <p>Name = Clear Alarm</p> <p>{default}</p> <p>Critical Temperature</p>
<p>Name *</p> <p>On Alarm</p> <hr/> <p>Transform</p> <p>function Transform(msg, metadata, msgType) {</p> <pre> 1 msg.xAlarm = 1; 2 return {msg: msg, metadata: metadata, msgType: msgType}; </pre> <p>msg.xAlarm = 1;</p> <p>return {msg: msg, metadata: metadata, msgType: msgType};</p>	<p>Add (4/6)</p> <p>Transformation Script</p> <p>Name = On Alarm</p>
<p>Name *</p> <p>Off Alarm</p> <hr/> <p>Transform</p> <p>function Transform(msg, metadata, msgType) {</p> <pre> 1 msg.xAlarm = 0; 2 return {msg: msg, metadata: metadata, msgType: msgType}; </pre> <p>msg.xAlarm = 0;</p> <p>return {msg: msg, metadata: metadata, msgType: msgType};</p>	<p>Add (5/6)</p> <p>Transformation Script</p> <p>Name = Off Alarm</p>
<p>Action - save timeseries</p> <hr/> <p>Details Events Help</p> <hr/> <p>Name *</p> <p>save TS</p>	<p>Add (6/6)</p> <p>Save time series</p> <p>Name = save TS</p>



4.2 Forward data from root rule chain

<p>Rule Chain</p> <p>rule chain</p>	<p>Edit Root Rule Chain</p> <p>Add rule chain</p>
<p>Open rule chain</p> <p>Rule chain *</p> <p>pkT423_Set&Clear Alarm</p> <p>pkT423_Set&Clear Alarm</p> <p>Description</p>	
<pre> graph LR Input[Input] --> DeviceProfile[device profile Device Profile Node] DeviceProfile -- Success --> MessageSwitch[message type switch Message Type Switch] MessageSwitch -- Post attributes --> SaveAttributes[save attributes Save Client Attributes] MessageSwitch -- Post telemetry --> SaveTS[save timeseries Save Timeseries] MessageSwitch -- RPC Request from Device --> LogRPC[log Log RPC from Device] MessageSwitch -- Other --> LogOther[log Log Other] MessageSwitch -- RPC Request to Device --> RPCRequest[rpc call request RPC Call Request] SaveAttributes --> Success1[Success] SaveTS --> Success2[Success] Success1 --> RuleChain[rule chain pkT423_Set&Clear AL...] Success2 --> RuleChain </pre> <p>The flowchart starts with an 'Input' node leading to a 'device profile Device Profile Node'. A 'Success' condition leads to a 'message type switch Message Type Switch' node. This switch node branches into five paths: 'Post attributes' (leading to 'save attributes Save Client Attributes'), 'Post telemetry' (leading to 'save timeseries Save Timeseries'), 'RPC Request from Device' (leading to 'log Log RPC from Device'), 'Other' (leading to 'log Log Other'), and 'RPC Request to Device' (leading to 'rpc call request RPC Call Request'). The 'Post attributes' and 'Post telemetry' paths converge into a 'Success' node, which then leads to a final 'rule chain pkT423_Set&Clear AL...' node.</p>	

5. Create Dashboard

<div data-bbox="203 247 594 338">Add Dashboard</div> <div data-bbox="232 375 594 447"> <p>Title *</p> <p>pkT423_Set&Clear Alarm</p> </div>	<p>Name = pkT423_Set&Clear Alarm</p>								
<div data-bbox="203 491 889 590">Add alias ×</div> <div data-bbox="224 611 889 877"> <div data-bbox="224 611 522 684"> <p>Alias name *</p> <p>pkT423_Set&Clear Data</p> </div> <div data-bbox="646 625 859 688"> <p>Resolve as multiple entities</p> <p><input type="checkbox"/></p> </div> <div data-bbox="224 705 859 779"> <p>Filter type *</p> <p>Single entity</p> </div> <div data-bbox="224 800 859 877"> <p>Type *</p> <p>Device</p> <div data-bbox="406 800 786 877"> <p>Device *</p> <p>pk_T423_BoundaryCheckData</p> <p>×</p> </div> </div> </div> <div data-bbox="690 989 881 1041"> <p>Cancel</p> <p>Add</p> </div>	<p>Edit Mode</p> <p>Add Alias</p>								
<div data-bbox="203 1073 657 1146"> <p>← Alarm widgets: select widget</p> </div> <div data-bbox="224 1167 657 1339"> <table border="1"> <thead> <tr> <th>Type</th> <th>Severity</th> </tr> </thead> <tbody> <tr> <td>Temperature</td> <td>Major</td> </tr> <tr> <td>Temperature</td> <td>Critical</td> </tr> <tr> <td>Low Humidity</td> <td>Warning</td> </tr> </tbody> </table> <div data-bbox="427 1167 610 1297"> <p>Alarms table</p> <p>Alarm widget</p> <p>Displays alarms based on defined time window and other filters.</p> </div> </div>	Type	Severity	Temperature	Major	Temperature	Critical	Low Humidity	Warning	<p>Add Alarm widget</p>
Type	Severity								
Temperature	Major								
Temperature	Critical								
Low Humidity	Warning								
<div data-bbox="224 1377 781 1705"> <p><input type="checkbox"/> Search propagated alarms</p> <div data-bbox="256 1461 399 1486"> <p>Alarm source</p> </div> <div data-bbox="243 1535 436 1596"> <p>Entity</p> </div> <div data-bbox="243 1625 576 1705"> <p>Entity alias *</p> <p>pkT423_Set&Clear Data</p> <p>×</p> </div> </div>									

TEMP
38

0100

Horizontal bar

Latest values

Preconfigured gauge to display any value reading as a horizontal bar. Allows to configure value range, gradient colors and other settings.

Add Horizontal bar

Entity

Entity alias *

pkT423_Set&Clear Data x

Filter

= ● ~ xAlarm: xAlarm ✎ ✕

• Over 80°C

New Timeseries Line Chart

avg
43.03

temperature

14758

-60

temperature

XALARM

1.00

1

0

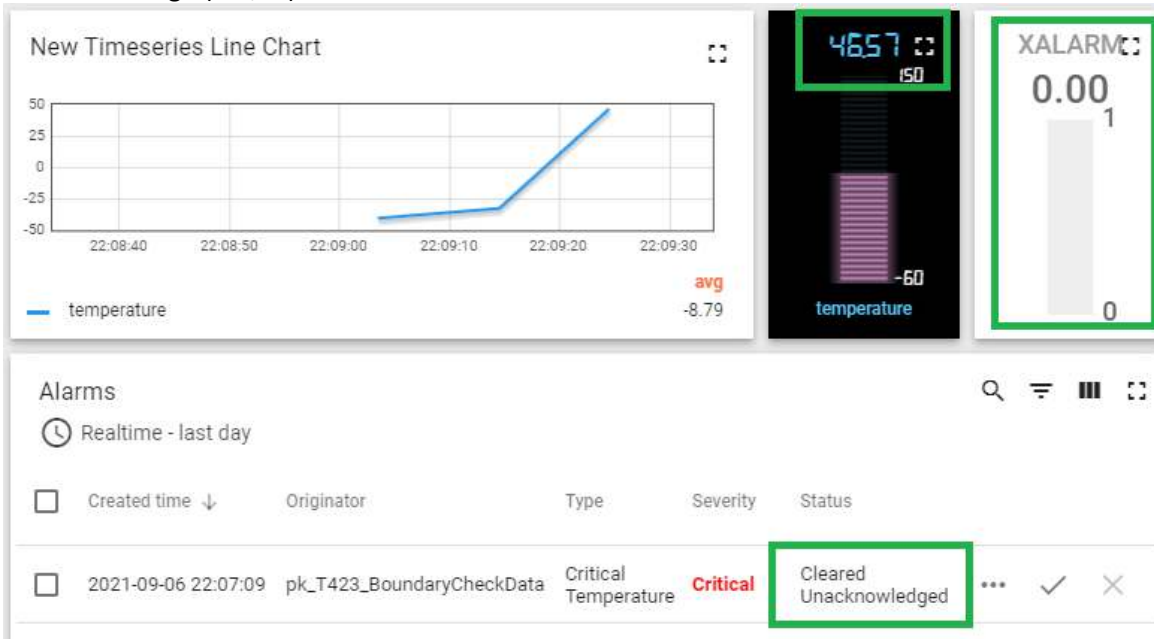
Alarms

🔍 ☰ ☷ 🗑

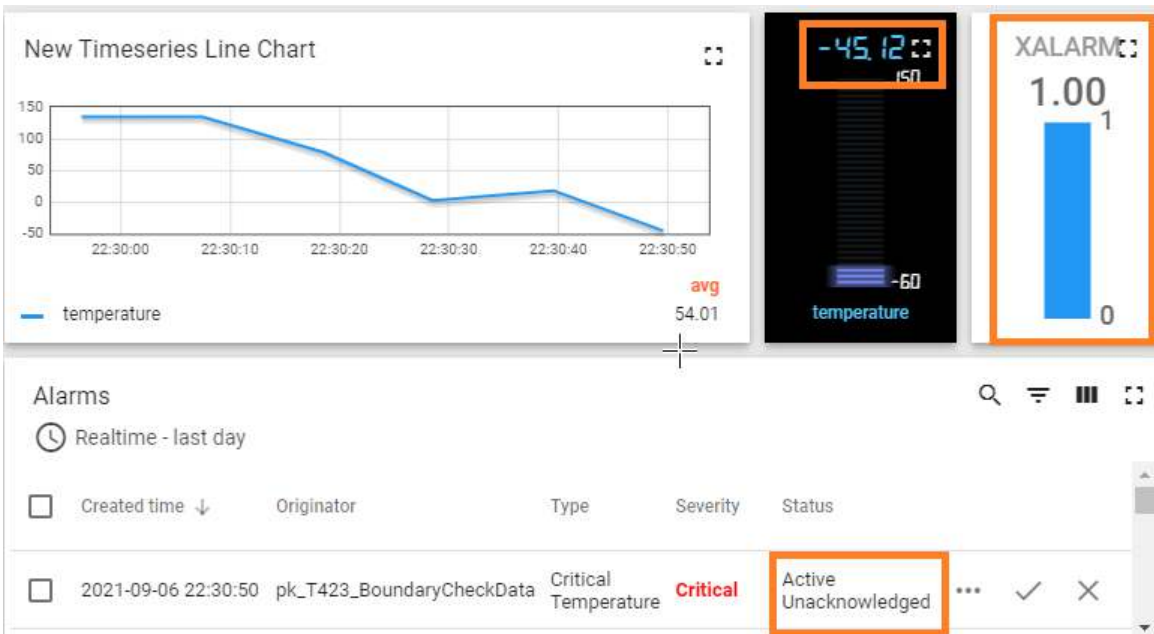
⌚ Realtime - last day

<input type="checkbox"/>	Created time ↓	Originator	Type	Severity	Status	
<input type="checkbox"/>	2021-09-06 22:11:12	pk_T423_BoundaryCheckData	Critical Temperature	Critical	Active Unacknowledged	... ✓ ✕

- In range (-40,80)



- Under -40°C



6. Test with curl

<https://reqbin.com/curl>

Curl

Raw

US

Run

```
curl -v -X POST -d '{"temperature":180}'
http://demo.thingsboard.io/api/v1/J0sCSdsE0MjElbKIbuUE/telemetry --header "Content-Type:application/json"
```

```
curl -v -X POST -d '{"temperature":180}'
```

```
http://demo.thingsboard.io/api/v1/A3MMKatweoQ4IpdsMFr2/telemetry --header "Content-Type:application/json"
```

Title *

pkT423_Set&Clear Alarm

Alarms



Realtime - last 2 minutes

<input type="checkbox"/>	Created time ↓	Originator	Type	Severity	Status
<input type="checkbox"/>	2021-09-05 16:23:57	pk_T423_BoundaryCheckData	Critical Temperature	Critical	Active Unacknowledged
<input type="checkbox"/>	2021-09-05 16:23:45	pk_T423_BoundaryCheckData	Critical Temperature	Critical	Cleared Unacknowledged
<input type="checkbox"/>	2021-09-05 16:23:00	pk_T423_BoundaryCheckData	Critical Temperature	Critical	Cleared Unacknowledged

Set Alarm

Clear Alarm

Lab403 – Create Alarm when the Device is offline

<https://thingsboard.io/docs/user-guide/rule-engine-2-0/tutorials/create-inactivity-alarm/>

This tutorial is to show you how to create an alarm when the device is offline for a certain period of time using RuleEngine.

1. Use Case

Let's assume the following use case:

- you have a device connected to ThingsBoard and this device has a temperature sensor to collect and push the telemetry data.
- the temperature sensor may stop pushing the telemetry data due to any kind of faults.

Therefore, in this case, you will need to configure ThingsBoard Rule Engine to:

- create an alarm if the device remains inactive for a certain period of time. This period of time can be defined in either of two ways:
 - The first way: by changing the global configuration parameter for the inactivity timeout. This parameter is defined in **thingsboard.yml** (**state.defaultInactivityTimeoutInSec**) and by default it is set to 10 seconds.
 - The second way: by overwriting this parameter for a particular device by setting the “**inactivityTimeout**” server-side attribute (value is set in milliseconds). This way will be described in the following sections.
- clear the alarm if the device becomes active.

2. Background

The ThingsBoard Device State service is responsible for monitoring the device connectivity state and triggering the device connectivity events that are pushed to Rule Engine.

ThingsBoard supports four types of events:

Event Type	Description
Connect	triggered when the device connects to ThingsBoard.
Disconnect	triggered when the device disconnects from ThingsBoard.
Activity	triggered when the device pushes a telemetry, an attribute update or RPC command.
Inactivity	triggered when the device is inactive for a certain period of time.

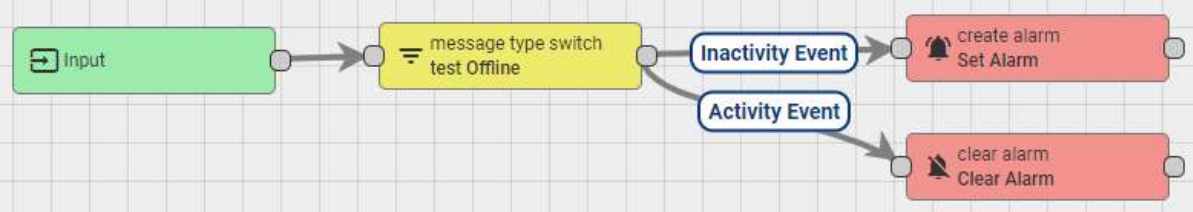
This tutorial will explain in details the device Inactivity event and it will show you how to:

- create Inactivity alarms using Rule Engine.
- configure a parameter for the inactivity timeout.

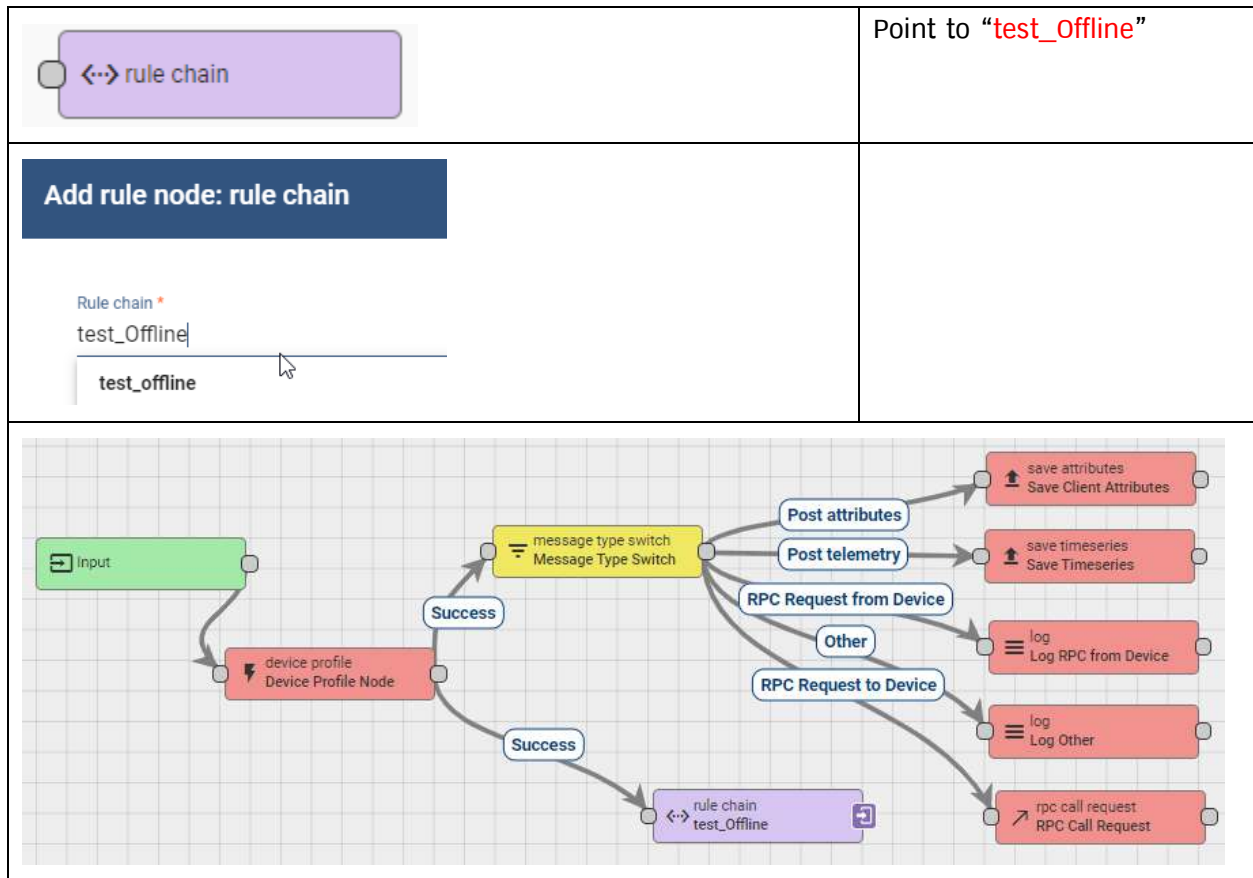
3. Adding the Device

<div>Add new device</div> <div>1 Device details</div> <div>Name * test_DeviceOffline</div>	Name = Test_DeviceOffline Copy Token ZNU1lVZtY6oy4JF0jDxD
<div>Details Attributes Latest teleme</div> <div> <div>Server attributes</div> <div>Entity attributes scope</div> <div>Server attributes</div> </div> <div> <input type="checkbox"/> Last update time Key ↑ </div>	Device Attributes Server attributes Add +
<div>Add attribute ✕</div> <div>Key * inactivityTimeout</div> <div>Value type Integer value *</div> <div>123 Integer 60000</div> <div>Cancel Add</div>	Add attribute Key = inactivityTimeout Integer 60000 milliseconds
<div>Server attributes</div> <div>Entity attributes scope</div> <div>Server attributes</div> <div> <input type="checkbox"/> Last update time Key ↑ Value </div> <div> <input type="checkbox"/> 2021-09-06 16:37:30 active false </div> <div> <input type="checkbox"/> 2021-09-06 16:42:56 inactivityTimeout 60000 </div>	

4. Configuring Rule Chains

<div>Add new device</div> <div>1 Device details</div> <div>Name *</div> <div>Test_DeviceOffline</div>	Add rule Chain Name = Test_Offline
<div>message type switch</div>	Name = test Offline
<div>create alarm</div>	Name = Set Alarm
<div>Alarm type *</div> <div>Slave_Offline</div> <div>Hint: use \${metadataKey} for value from metadata, \${message}</div> <div>Alarm severity *</div> <div>Critical</div> <div><input checked="" type="checkbox"/> Propagate</div>	Alarm Type = Salve_Offline Critical Propagate
<div>clear alarm</div>	Name = Clear Alarm
<div>Alarm type *</div> <div>Slave_Offline</div> <div>Hint: use \${metadataKey} for value from metadata,</div>	Alarm Type = Slave_Offline
	

Root Rule Chain




5. Monitor Alarm

Check Device → Alarm

- Active Slave_Offline

Details Attributes Latest telemetry **Alarms** Events Relations Audit Logs

Alarm status
Any  last 30 minutes

Created time ↓	Originator	Type	Severity	Status
2021-09-06 21:23:45	Test_DeviceOffline	Slave_Offline	Critical	Active Unacknowledged

• Pushing Data to Device → Clear Slave_Offline

2021-09-06 21:25:45	Test_DeviceOffline	Slave_Offline	Critical	Active Unacknowledged
2021-09-06 21:23:45	Test_DeviceOffline	Slave_Offline	Critical	Cleared Unacknowledged
2021-09-06 21:16:45	Test_DeviceOffline	Slave Offline - 1234	Critical	Active Unacknowledged

6. Adding Dashboard

- Alarm Table Widget
- Digital Horizontal bar Widget

Alarms
Realtime - last day

<input type="checkbox"/>	Created time ↓	Originator	Type	Severity	Status			
<input type="checkbox"/>	2021-09-06 21:30:45	Test_DeviceOffline	Slave_Offline	Critical	Active Unacknowledged	...	✓	×
<input type="checkbox"/>	2021-09-06 21:25:45	Test_DeviceOffline	Slave_Offline	Critical	Cleared Unacknowledged	...	✓	×
<input type="checkbox"/>	2021-09-06 21:23:45	Test_DeviceOffline	Slave_Offline	Critical	Cleared Unacknowledged	...	✓	×
<input type="checkbox"/>	2021-09-06 21:16:45	Test_DeviceOffline	Slave Offline - 1234	Critical	Active Unacknowledged	...	✓	×

Items per page: 10 1 - 9 of 9 |< < > >|

TEMPERATURE

67.14

100

0

7. Test – with <https://reqbin.com/curl>

```
curl -v -X POST -d '{"temperature":45.67}'
http://demo.thingsboard.io/api/v1/cgg5nawrMDAsFqwsPjOP/telemetry --header "Content-Type:application/json"
```

```
curl -v -X POST -d '{"temperature":45.67}'
http://demo.thingsboard.io/api/v1/cgg5nawrMDAsFqwsPjOP/telemetry --header "Content-Type:application/json"
```

7. Test – with Python

```
1 # Fix Data Send via Telemetry
2 import requests
3 from requests.structures import CaseInsensitiveDict
4
5 url = "http://demo.thingsboard.io/api/v1/cgg5nawrMDAsFqwsPjOP/telemetry"
6
7 headers = CaseInsensitiveDict()
8 headers["Content-Type"] = "application/json"
9 data = '{"temperature":55.66}'
10 resp = requests.post(url, headers=headers, data=data)
11
12 print(resp.status_code)
13
```

```
import requests
from requests.structures import CaseInsensitiveDict
url = "http://demo.thingsboard.io/api/v1/cgg5nawrMDAsFqwsPjOP/telemetry"
headers = CaseInsensitiveDict()
headers["Content-Type"] = "application/json"
data = '{"temperature":55.66}'
resp = requests.post(url, headers=headers, data=data)
print(resp.status_code)
```

```

1 # Random Data Send via Telemetry
2 import json, random, requests
3 from requests.structures import CaseInsensitiveDict
4
5 tdata = str(round(random.random() * 100,2))
6 data_to_send = {"temperature":tdata}
7 print('tdata = ',tdata)
8
9 url = "http://demo.thingsboard.io/api/v1/cgg5nawrMDAsFqwsPjOP/telemetry"
10 headers = CaseInsensitiveDict()
11 headers["Content-Type"] = "application/json"
12 resp = requests.post(url, headers=headers, data=json.dumps(data_to_send))
13 print('response.status_code = ',resp.status_code)
14
tdata = 97.96
response.status_code = 200

```

```

import json, random, requests
from requests.structures import CaseInsensitiveDict
tdata = str(round(random.random() * 100,2))
data_to_send = {"temperature":tdata}
print('tdata = ',tdata)
url = "http://demo.thingsboard.io/api/v1/cgg5nawrMDAsFqwsPjOP/telemetry"
headers = CaseInsensitiveDict()
headers["Content-Type"] = "application/json"
resp = requests.post(url, headers=headers, data=json.dumps(data_to_send))
print('response.status_code = ',resp.status_code)

```

8. Note

```

curl -v -X POST -d '{"temperature":60.1234}'
http://demo.thingsboard.io/api/v1/nM8fAhWbTMaQmBd5yxGr/attributes --header "Content-
Type:application/json"

```

```

curl -v -X POST -d '{"temperature":60.1234}'
http://demo.thingsboard.io/api/v1/nM8fAhWbTMaQmBd5yxGr/telemetry --header "Content-
Type:application/json"

```

```

curl -v -X POST -d '{"temperature":60.1234}'
203.158.12.34:8080/api/v1/nM8fAhWbTMaQmBd5yxGr/telemetry --header "Content-
Type:application/json"

```

3/5 -- Case Study 1 - ตัวอย่างการสร้างระบบตรวจสอบและควบคุมอุปกรณ์

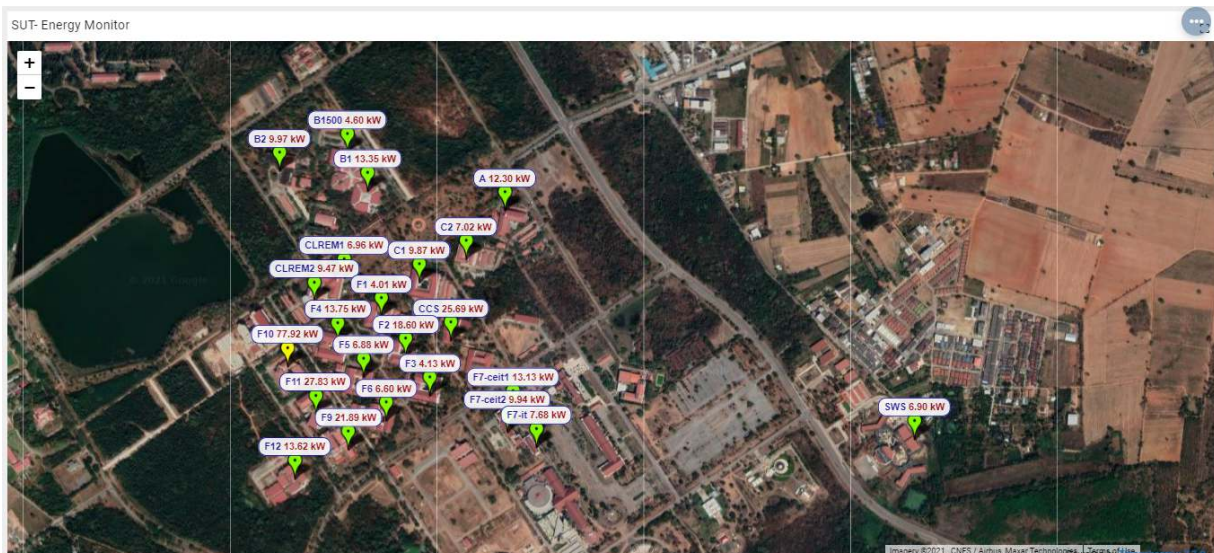
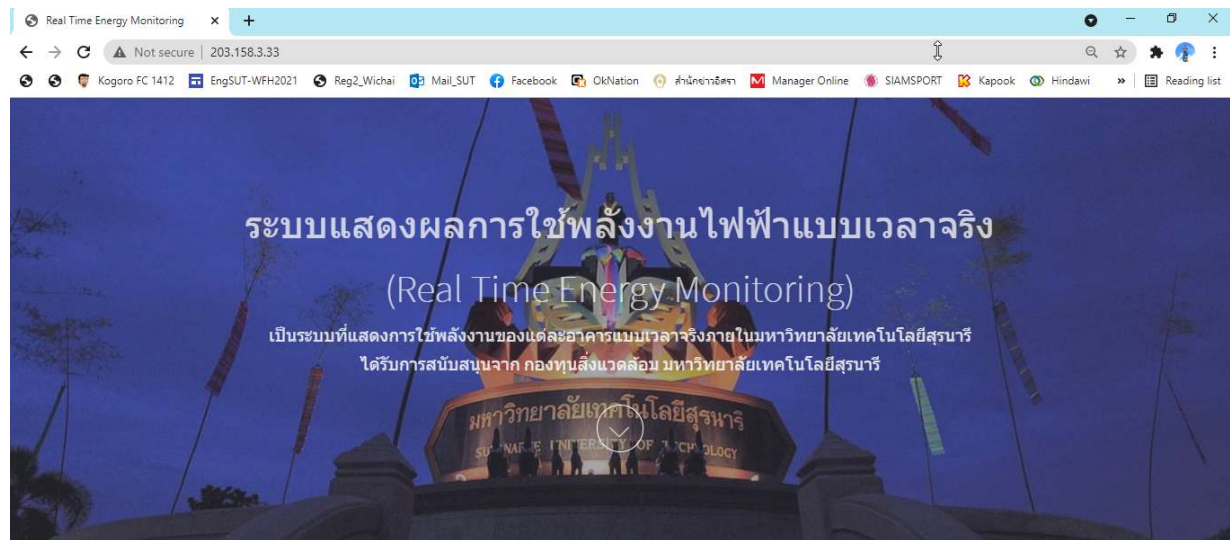
Lab404 -- XXXXXXXX

< Skip >

4/5 -- Case Study 2 – ตัวอย่างการสร้างระบบตรวจสอบการใช้พลังงาน

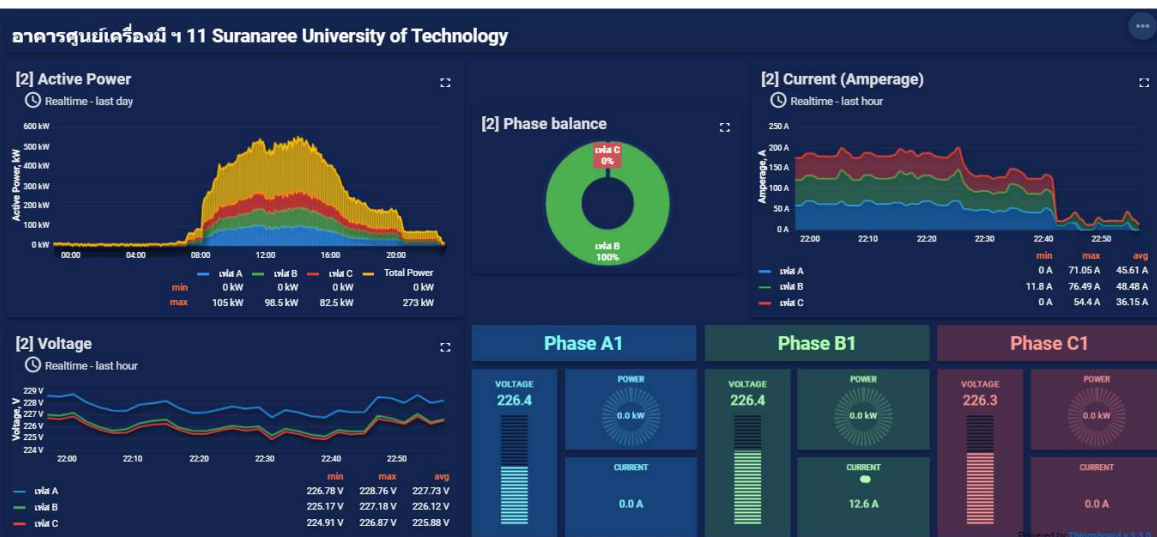
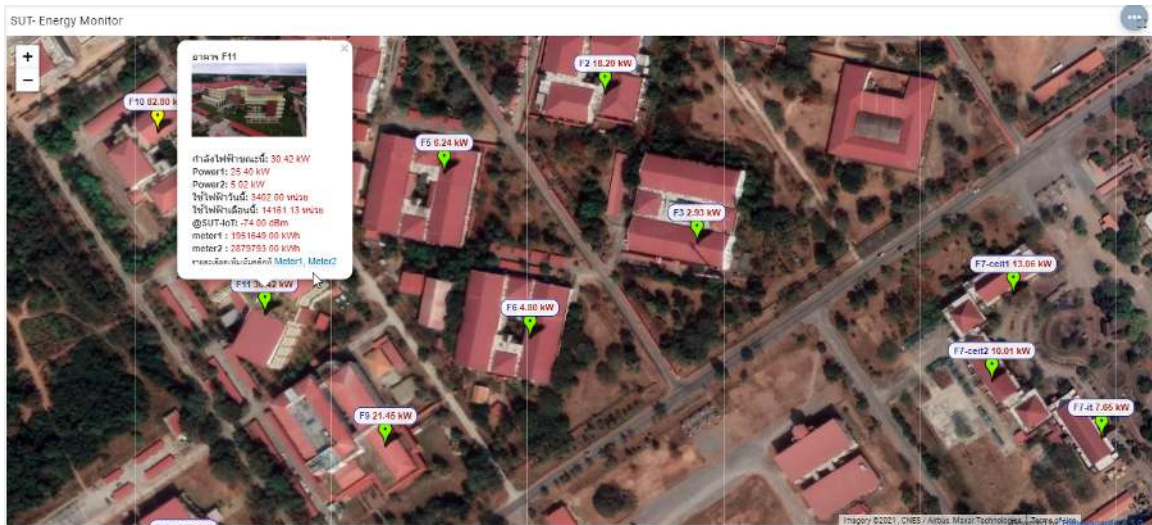
1. ระบบแสดงผลการใช้พลังงานไฟฟ้าแบบเวลาจริง (Real Time Energy Monitoring)

- ระบบที่กำลังดำเนินงานและอยู่ระหว่างขยายจำนวนทดสอบ
- Sever IP = 203.158.3.33
- จำนวนกลุ่มอาคารทดสอบ = 23 กลุ่มอาคาร
- จำนวน Digital Power Meter = 25 ตัว
- ระบบสื่อสารผ่าน SUT - Wifi และ SUT - LoRa WAN
- หน้าหลักสำหรับรายงานผล





• กลุ่มอาคารศูนย์เครื่องมือฯ 11



2. Digital Power Meter

<https://my.factomart.com/products/power-distribution/metering/digital-power-meter>

ปัจจุบันอุตสาหกรรมในประเทศไทยได้พัฒนาไปอย่างรวดเร็วจึงส่งผลให้มีการใช้ พลังงานในรูปแบบต่างๆ เพิ่มขึ้นทุกปี ด้วยเหตุนี้จึงจำเป็นต้องมีการบริหารจัดการพลังงานมาให้เพียงพอและเหมาะสม การจัดการพลังงานก็มีหลากหลายรูปแบบ เช่น การปรับปรุงค่าตัวประกอบทางไฟฟ้า Power Factor, การจัดการและบริหารการใช้ไฟฟ้า (Demand Controller) มีต้นทุนที่ต่ำสุดจึงเป็นที่นิยมมากที่สุด, การเปลี่ยนไปใช้ Frequency Converter สำหรับงานลักษณะการควบคุมการไหลของของเหลว, การเลือกใช้เครื่องใช้ไฟฟ้าที่มีประสิทธิภาพสูง (เครื่องใช้ไฟฟ้า หรืออุปกรณ์ที่ทำงานมากขึ้นแต่กินไฟฟ้าน้อยลง) เป็นต้น

ซึ่งการจัดการและบริหารไฟฟ้า หรือที่เรียกว่าการควบคุม Demand เป็นวิธีการที่มีต้นทุนที่ต่ำจึงเป็นที่นิยมมากที่สุดโดยสามารถนำ Power Meter มาช่วยในการจัดการได้ดังนี้

2.1 การนำไปใช้ในการบริหารจัดการพลังงาน โดยผ่านระบบซอฟต์แวร์คอมพิวเตอร์

การนำข้อมูลจาก Power Meter เข้าสู่ระบบซอฟต์แวร์ของคอมพิวเตอร์เพื่อใช้ในการบริหารจัดการพลังงาน โดยผ่านซอฟต์แวร์สำหรับการบริหารจัดการ และควบคุมพลังงานไฟฟ้า เพื่อให้ผู้ใช้ไฟฟ้าสามารถบริหารและวางแผนการใช้ไฟฟ้าอย่างมีประสิทธิภาพสูงสุด โดยสามารถดูข้อมูลต่างๆ ได้แบบ Real time และ สามารถแสดงค่าต่างๆ ในรูปแบบกราฟได้อีก

2.2 ควบคุมค่าดีมานด์อัตโนมัติ

คือ การนำค่าเฉลี่ยที่สูงสุดในรอบเดือนมาใช้เป็นค่าความต้องการใช้ไฟฟ้าสูงสุด (Maximum Demand) ดังนั้นถ้าค่า Demand มีแนวโน้มที่สูงเกินไปควรที่จะเลื่อนการทำงานบางอย่างที่ไม่จำเป็นต้องออกไปก่อนหรือตัด Load หรือเครื่องใช้ไฟฟ้าที่สามารถหยุดได้ก่อน อาทิเช่น เครื่องปั๊มน้ำ เครื่องทำความเย็น ระบบปรับอากาศ หรือเครื่องจักรที่สามารถหยุดได้ชั่วขณะเป็นต้นเพื่อหลีกเลี่ยงการเสียค่าไฟสูงในเดือนนั้นๆ

2.3 ตรวจเช็คค่าต่างๆ และคุณภาพของระบบไฟฟ้า

สามารถใช้ในการเรียกดูข้อมูลเป็นตัวเลขแบบ Real Time ในรูปแบบตารางพร้อมกันทุกมิเตอร์ในระบบ

2.4 คัด Billing ค่าไฟฟ้าแยกส่วนต่างๆ

เช่น ในห้องพักโรงแรม/รีสอร์ท ร้านค้าเช่า ส่วนจัดแสดงสินค้าหรือบูธ รวมไปถึงสถานที่ จัดงานที่ คัดค่าไฟชั่วคราว เป็นต้น

2.5 สามารถนำไปใช้งานในฟังก์ชันอื่นๆเพิ่มเติม

เช่น การทำ Alarm Control , การรับ-ส่งสัญญาณ Digital/Analog เป็นต้น

Power Meter คือ อุปกรณ์ที่จะช่วยบอกค่าทางไฟฟ้าและสามารถช่วยควบคุมการทำงานต่างๆได้ อีกทั้งยังสามารถทำงานร่วมกับระบบซอฟต์แวร์คอมพิวเตอร์อื่นๆอย่างหลากหลายซึ่งสามารถเพิ่มความสะดวกสบายในการทำงาน และมีความจำเป็นอย่างมากในงานของภาคอุตสาหกรรม

3. การสื่อสารกับ Power meter



รูปแบบการสื่อสารกับ Power Meter ที่นิยมใช้ในปัจจุบัน

3.1 รูปแบบการสื่อสาร RS-485

รูปแบบการสื่อสาร RS-485 เป็นการสื่อสารที่นิยมใช้กันมากในงานระบบสื่อสารโรงงาน ตึกอาคารพาณิชย์ จุดเด่น RS-485

- ส่งสัญญาณได้ไกล
RS485 สามารถส่งสัญญาณได้ไกลสูงสุดถึง 1,200 เมตร ซึ่งถือว่าเป็นระยะทางที่ไกลมาก เพียงพอต่อการใช้งานในโรงงานอุตสาหกรรมอย่างแน่นอนและจะเห็นได้ชัดว่าระยะการส่งสัญญาณได้ถูกพัฒนาขึ้นมากจนทั้งห่างมาตรฐานรุ่นเก่าอย่าง RS232 ที่สามารถส่งสัญญาณได้เพียง 15 เมตร เท่านั้น
- เชื่อมต่อเครือข่ายได้
นอกจากจะส่งสัญญาณได้ไกลแล้ว RS485 ยังสามารถเชื่อมต่อเป็นเครือข่าย (Network) แบบ Multipoint ได้ด้วย ซึ่งสามารถเชื่อมต่ออุปกรณ์ในระบบได้สูงสุดถึง 32 ตัว ซึ่งสิ่งนี้ถือว่าเป็นอีกหนึ่งจุดเด่นของสัญญาณ RS485 เลยทีเดียว
- ช่วยประหยัดงบประมาณในการเดินสาย
RS485 เป็นมาตรฐานที่ใช้สายไฟเพียง 2 เส้นในการรับส่งข้อมูล เมื่อเปรียบเทียบกับมาตรฐานรุ่นเก่าที่สามารถส่งสัญญาณในระยะเท่ากันอย่าง RS422 ที่ต้องใช้สายไฟถึง 4 เส้นในการรับส่งข้อมูล ซึ่งราคาสายเคเบิลแบบ 2 แกน จะถูกกว่าสายเคเบิลแบบ 4 แกน ถึงเกือบครึ่ง ในความเป็นจริงแล้วเรื่องงบประมาณถือเป็นเรื่องสำคัญมากๆ ซึ่งนี่ถือเป็นอีกหนึ่งจุดเด่นของ RS485 เลยทีเดียว

ข้อเสีย RS-485

- ต้องใช้ตัวแปลงสัญญาณในการเชื่อมต่อกับคอมพิวเตอร์
เนื่องจากปัจจุบันคอมพิวเตอร์ที่เราใช้กันอยู่นั้นไม่มี port เชื่อมต่อสัญญาณ RS485 โดยตรง จะมีก็แค่ USB หรือ RS232 เท่านั้น ฉะนั้นหากเราจะเชื่อมต่ออุปกรณ์ที่ใช้ RS485 กับคอมพิวเตอร์นั้น เราต้องเสียงบประมาณเพิ่มขึ้นในการซื้อตัวแปลงสัญญาณ (Converter) เพื่อแปลงสัญญาณจาก RS485 เป็น USB หรือ RS232 ในการเชื่อมต่อนั่นเอง
- ความเร็วในการรับส่งข้อมูลช้า
ถึงแม้ RS485 จะถูกพัฒนาด้านความเร็วในการรับส่งข้อมูลขึ้นมากแล้วก็ตามเมื่อเทียบกับมาตรฐานเก่า แต่ก็ยังมีความล่าช้าอยู่เมื่อเชื่อมต่อในลักษณะเครือข่ายจำนวนมากๆ

3.2 รูปแบบการสื่อสาร Ethernet

เป็นการสื่อสารที่ได้รับความนิยมในการใช้งานการเข้าถึงสื่อสารสนเทศมากที่สุด ซึ่งสามารถเชื่อมต่อกับ Power Meter ได้เช่นกัน

จุดเด่น Ethernet

- ความรวดเร็วในการรับส่งข้อมูล
มีความเร็วในการรับส่งข้อมูลที่สูงมาก รวมถึงรูปแบบการรับ-ส่งข้อมูลมีให้เลือกใช้งานในแบบ Full-Duplex ทำให้ความสามารถในการรับส่งข้อมูลได้ตอบโต้ได้เป็นอย่างดี
- ง่ายต่อการเข้าถึงและใช้งาน
เป็นรูปแบบที่นิยมมากที่สุดง่ายต่อการเข้าถึงและใช้งาน ไม่ต้องแปลงสัญญาณใดๆสามารถต่อสาย LAN แล้วเข้ากับคอมพิวเตอร์ได้ทันที

ข้อเสีย Ethernet

- ระยะการส่งมีจำกัด
เมื่อเทียบกับ RS-485 ระยะทางค่อนข้างสั้น คือสูงสุด 100 เมตรเท่านั้น ถ้าหากต้องใช้ระยะไกลต้องมีการทวนสัญญาณตลอดระยะทางทำให้สิ้นเปลืองค่อนข้างมาก
- ราคาสูง
ราคาของสาย LAN ที่ใช้ในการเชื่อมต่อมีราคาสูงเมื่อเทียบกับสาย 2 core แบบมีชีลด์ และเมื่อต้องต่อเข้าระบบจำนวนมากต้องใช้ Switch hub ในการเชื่อมต่อ ซึ่งมีราคาค่อนข้างสูงเมื่อเทียบกับจำนวนพอร์ตการเชื่อมต่อ

3.3 รูปแบบการสื่อสาร USB

เป็นรูปแบบการสื่อสารที่ง่าย เพียงแค่ต่อสาย USB ก็สามารถต่อเข้ากับ PC Notebook ได้เลยไม่ต้องผ่านการแปลง

จุดเด่น USB

- ใช้งานง่าย
ง่ายต่อการใช้งานเพียงแค่ต่อสาย USB ก็สามารถต่อเข้ากับคอมพิวเตอร์ได้เลย
- ความรวดเร็วในการรับส่งข้อมูล
มีความเร็วสูงรับ-ส่งข้อมูล ตอบสนองทันที

ข้อเสีย USB

- ระยะทางเชื่อมต่อสั้น
ระยะทางในการเชื่อมต่อ USB นั้นสั้นมาก การเชื่อมต่อที่เสถียรที่สุดคือ 5 เมตร
- ติดตั้ง Driver USB
ในการใช้งานผ่านทาง USB โดยทั่วไปต้องมีการติดตั้ง Driver USB ด้วย อาจทำให้ไม่สะดวกในบางกรณี

Modbus Protocol ที่ใช้ในงาน Power Meter

หลังจากที่เราทราบแล้วว่ารูปแบบการสื่อสารที่นิยมใช้ส่วนใหญ่เป็นรูปแบบไหน เรามาดูกันต่อว่า Modbus Protocol ที่ Power Meter ส่วนใหญ่ใช้มีอะไรบ้าง

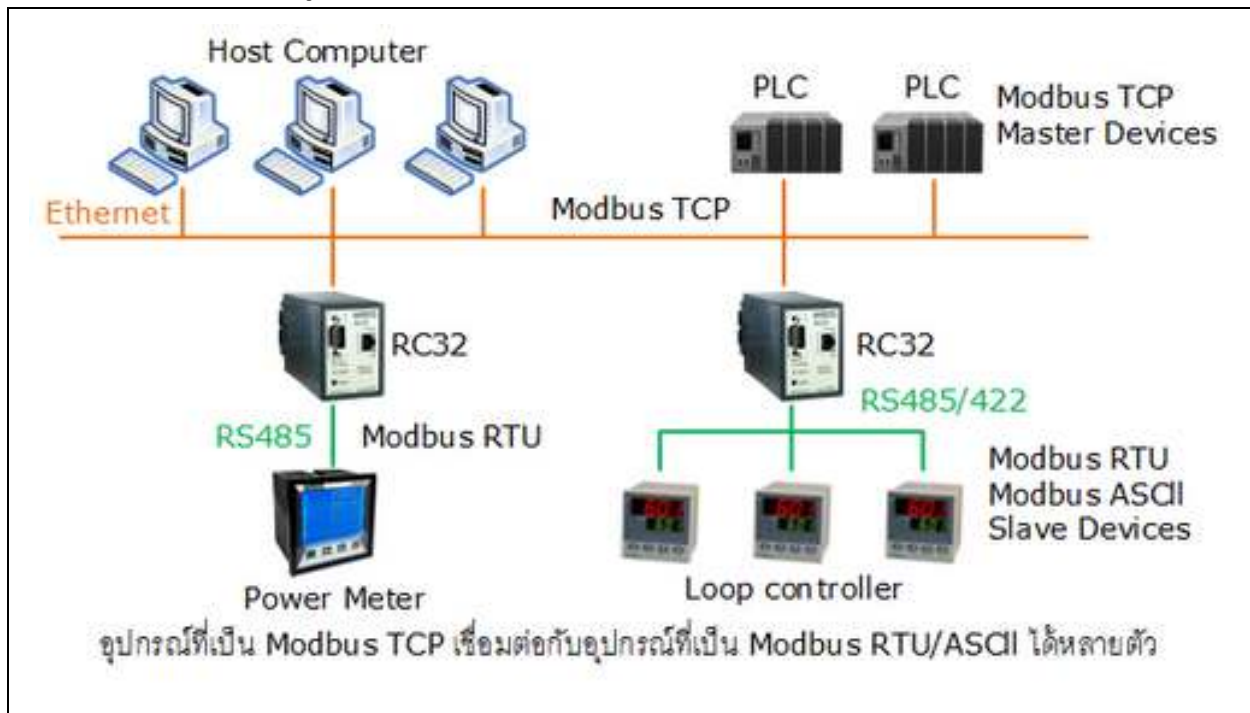
- **Modbus RTU**

ใช้ในการสื่อสารแบบอนุกรมและทำให้การใช้งานมีขนาดกะทัดรัดแทน binary ของข้อมูลสำหรับโปรโตคอลการสื่อสาร รูปแบบ RTU ตามคำสั่งข้อมูลที่มีวงจรตรวจสอบความซ้ำซ้อนการตรวจสอบเป็นกลไกการตรวจสอบข้อผิดพลาดเพื่อความน่าเชื่อถือของข้อมูล

- **Modbus TCP/IP (Modbus-TCP)**

คือ โปรโตคอล Modbus RTU ที่เชื่อมต่อกับ TCP โดยทำงานบนสถาปัตยกรรมของ Ethernet โครงสร้าง Message ของ Modbus คือ application protocol ที่จะถูกส่งผ่านไปพร้อมกับ TCP/IP (TCP/IP คือ Transmission Control Protocol และ Internet Protocol ซึ่งเป็นตัวกลางที่ใช้ในการส่ง Message ของ Modbus TCP/IP)

ซึ่งทั้ง 2 รูปแบบ สามารถใช้งานในระบบ Power Meter ได้ทั้งสองแบบขึ้นกับระบบอุปกรณ์ที่ใช้งานรวมเข้ากับระบบ เช่น PLC HMI รองรับแต่ TCP ก็สามารถใช้งานในรูปแบบ TCP ที่วิ่งผ่าน RS-485 ได้เช่นกัน หรือ อุปกรณ์รองรับ RTU ก็สามารถใช้ในแบบ LAN หรือ USB ก็ได้

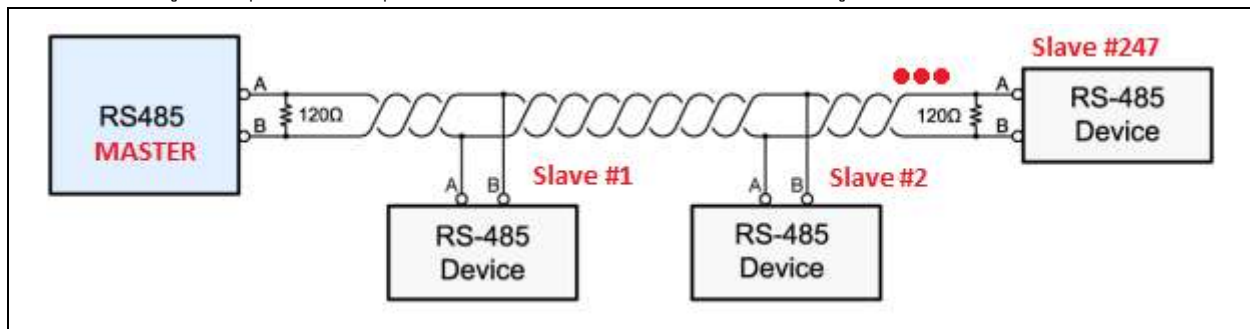


Lab405 – Read Data from Modbus RTU Device

1. การสื่อสารผ่าน Modbus Protocol

Modbus คือ โปรโตคอล (Protocol) การสื่อสารที่พัฒนาขึ้นโดย บริษัท Modicon Systems ด้วยรูปแบบง่ายๆ เป็นรูปแบบการส่งข้อมูลระหว่างอุปกรณ์อิเล็กทรอนิกส์ อุปกรณ์ที่ต้องการข้อมูลเรียกว่า Modbus Master (Client) ส่วนอุปกรณ์ที่ให้ข้อมูลที่ต้องการเรียกว่า Modbus Slave (Server) ใน Modbus Network ที่เป็นมาตรฐานนั้นจะมี Master ตัวเดียวแต่ Slave มีได้ถึง 247 ตัว โดยแต่ละตัวจะมี ID ระบุเหมือนเลขที่บ้านตั้งแต่ 1 ถึง 247 และ Master สามารถ Write ข้อมูลไปยัง Slave ได้

Modbus เป็น Open Protocol หมายความว่า บุคคลทั่วไปสามารถพัฒนาอุปกรณ์ที่ใช้การสื่อสารแบบ Modbus โดยไม่ต้องเสียค่าใช้จ่ายใดๆ Modbus จึงเป็น Protocol พื้นฐานและนิยมใช้อย่างแพร่หลายในทุกอุตสาหกรรม โดยรับส่งข้อมูลจากอุปกรณ์ควบคุมกับ Controller หรือระบบประมวลผลข้อมูลต่าง



แสดงการสื่อสารระหว่าง Master กับ Slave

Slave อาจเป็นอุปกรณ์ต่อพ่วงใดๆ เช่น Input/Output Transducer, วาล์ว (Valve), Inverter, อุปกรณ์บันทึกข้อมูล (Data Logger) หรืออุปกรณ์เครื่องมือวัดอื่นๆ เป็นต้น ซึ่งประมวลผลและส่งข้อมูลไปยัง Master

Master สามารถติดต่อกับ Slave แต่ละตัวได้หรือสามารถส่งเป็น Message ถึง Slave ทุกตัวได้ในลักษณะของการ Broadcast และ Slave จะตอบสนองสิ่งที่ Master ต้องการเท่านั้น สิ่งที่ Master ส่งให้จะประกอบด้วย Slave Address, Function Code (คำสั่งหรือสิ่งที่ต้องการให้ทำ), Data และ Checksum ส่วนข้อมูลที่ Slave ส่งกลับมาจะประกอบด้วยคำสั่งที่สั่งให้กระทำหรือข้อมูลต่างๆ และ Checksum

Modbus แบ่งออกเป็น Modbus Serial: ASCII/RTU (เป็นพอร์ตการสื่อสารแบบอนุกรม RS232, RS485, RS422) และ Modbus TCP/IP (LAN) เพื่อให้สอดคล้องกับแนวโน้มการพัฒนาการสื่อสารในปัจจุบันและทุกอย่างที่สามารถเชื่อมต่อกับเครือข่าย Ethernet หรือ Internet เพื่อส่งข้อมูล

- Modbus Serial เป็นการสื่อสารโดยการส่งข้อมูลไปตามสายสัญญาณ Serial ระหว่างอุปกรณ์ โดยวิธีการสื่อสารที่ง่ายที่สุดคือการต่อสายสัญญาณ Serial ระหว่าง Master หนึ่งตัวกับ Slave หนึ่งตัว (Point-to-Point) ซึ่งจะเป็นการสื่อสารผ่านพอร์ต RS232 หรือ RS422 หรือการต่อ Master หนึ่งตัวกับ Slave หลายตัว (Point-to-Multipoint) ซึ่งจะเป็นการสื่อสารผ่านพอร์ต RS485

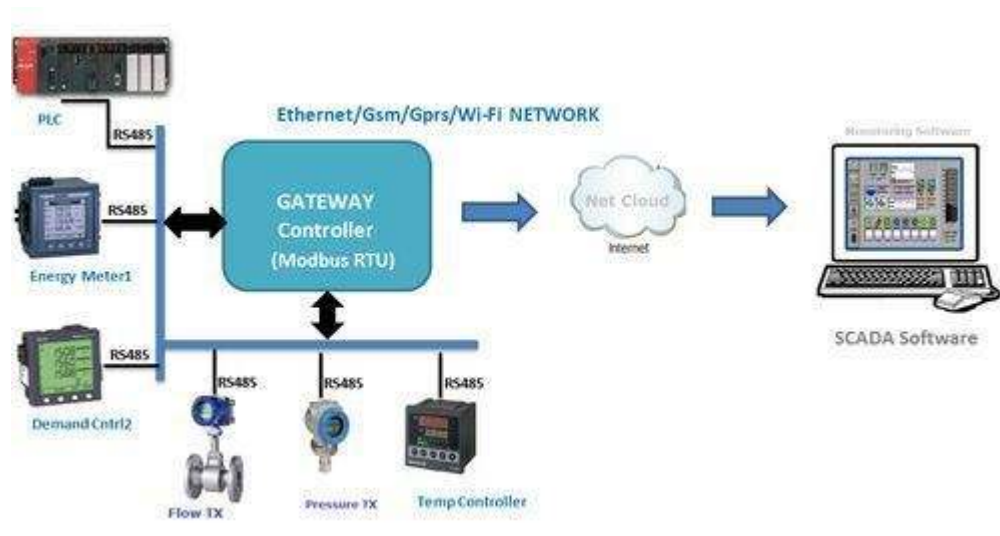
การรับส่งข้อมูลด้วยโปรโตคอล Modbus Serial สามารถเลือกได้ 2 โหมด คือ โหมด ASCII และ โหมด RTU ซึ่งทั้ง 2 โหมดนี้มีความแตกต่างกันที่การกำหนดรูปแบบของชุดข้อมูลภายในเฟรม จะเลือกโหมดใดก็ได้แต่มีเงื่อนไขว่าอุปกรณ์ทุกตัวที่ต้องร่วมกันอยู่ในบัสหรือเครือข่ายเดียวกัน จะต้องตั้งให้เลือกใช้โหมดเดียวกันทั้งหมด รวมถึง Serial Parameter ต่างๆ เช่น Baud Rate, Data Bit, Stop Bit และ Parity Bit

- Modbus TCP/IP ถูกพัฒนาขึ้นโดยมีวัตถุประสงค์เพื่อจะนำการสื่อสารแบบ Ethernet มาใช้กับอุปกรณ์จำพวก Ethernet Device ระยะในการใช้งานสำหรับการเดินสาย (สาย LAN) คือ 100 เมตร โดยสามารถขยายระยะในการสื่อสารได้โดยการใช้อุปกรณ์ Repeater หรือในระบบ LAN จะเรียกอุปกรณ์นี้ว่า Hub หรือ Switch ก็จะสามารถลากสายได้อีก 100 เมตร และยังสามารถต่อ Repeater ขยายระยะทางได้โดยไม่จำกัด ในการสื่อสารโดยทั่วไปมีความเร็ว 100,000,000 บิตต่อวินาที (100 Mbps) และเชื่อมต่ออุปกรณ์ได้ไม่จำกัดจำนวน

สำหรับอุปกรณ์ Modbus Serial ที่จะติดต่อสื่อสารกับอุปกรณ์ Modbus TCP/IP เพื่อให้ใช้งานในเครือข่าย Ethernet จะใช้ Gateway ติดต่อและแปลงรูปแบบการสื่อสารข้อมูล โดยการสื่อสารของ Modbus Serial จะถูก Gateway แปลงให้เป็น Modbus TCP/IP เพื่อใช้ในการติดต่อสื่อสารในเครือข่าย Ethernet

2. การแปลงระหว่าง Modbus RTU/TCP ด้วย Modbus Gateway

MODBUS GATEWAY เป็นอุปกรณ์ที่ทำหน้าที่เป็นตัวกลาง ในการเชื่อมต่ออุปกรณ์ที่ใช้การสื่อสารในรูปแบบ MODBUS ระหว่างการสื่อสารที่เป็น MODBUS TCP (ETHERNET LAN) และการสื่อสารที่เป็น UART ทั้งแบบ RS232 หรือ RS422/485 ให้สามารถเชื่อมต่อกันได้



ADM-5805G {ETT @ 1,890.00THB}



HF5111A {Lazada @959.00THB}

3. Digital Power Meter – EDMl Mk6E



The Mk6E is an enhanced upgrade of the Mk6 meter, built with a higher-class accuracy of 0.2S, catering to the high-end markets. The Mk6E is a high-precision meter created for generation and transmission applications, as well as for revenue metering at high-end consumer facilities.

- AMI Ready
- High Accuracy
- Large Data Storage
- Large LCD Display
- Upgradeable
- Anti Tamper
- Script Extensions

3.1 Read Data from Mk6E

	<p>MK6</p> <p>Baud rate (default)</p> <p>= 9600N81</p>
	<p>MK6E</p> <p>Baud rate (default)</p> <p>= 9600N81</p>

3.2 Protocol

- The protocol used is EDMI's command line protocol

Register	Register Number	Data Type	Example Value	Unit	Security Group
Phase A Voltage	E000	Float	237.345	Volts	35
Plant Number	F00D	String	Fred Electric	None	21
Number of Billing Resets	F032	Long	453	None	92
Last Billing Reset	FC00	Time/Date	14:30:24 28/8/98	None	93
Frequency	E060	Float	50.056	Hz	200
Channel 1 Unified Energy	0009	Double	12332543.12234	Wh	42

- Table 5-1 Example Registers

4. Digital Power Meter – Mitsubishi SX1-A31E

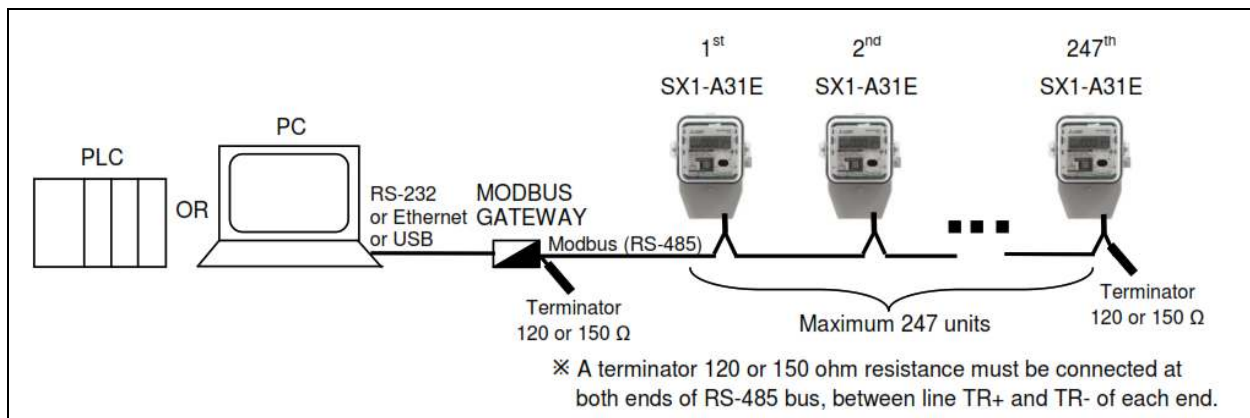
4.1 มิเตอร์ SX1: 1เฟส รุ่น RS-485 มิเตอร์อิเล็กทรอนิกส์ เอเอ็มอาร์



สเปคมิเตอร์ METER SPECIFICATIONS	
รหัสรุ่น	SX1-A31E
มาตรฐานStandard	มอก. 2543-2555, IEC 62052-11, IEC 62053-21
การแสดงผลDisplay	LCD
ค่าพลังงานไฟฟ้า (หน่วย)Energy (kWh)	5 หลักจำนวนเต็ม + 1 หลักทศนิยม
ค่าแสดงบนหน้าจอItem Display	kWh, Volt, Amp, kW
พอร์ตสื่อสารCommunication port	RS-485 (สำหรับเชื่อมต่อ AMR)
OPERATING CONDITION	
ระบบไฟPower System	1 เฟส 220-230 โวลต์
ช่วงแรงดันใช้งานOperation Voltage	176 - 264 โวลต์
กระแสฟักัดRated Current	5(45)A
ความถี่อ้างอิงReference Frequency	50 Hz
ช่วงอุณหภูมิ / ความชื้นTemperature / Humidity	0-70 °C / 0-98 %RH

ELECTRICAL CHARACTERISTICS	
ความแม่นยำการวัดAccuracy	Class 1
ค่าการกระพริบตัวอินดิเคเตอร์LCD Pulse	3000 ครั้ง/หน่วย, (3000 imp/kWh)
การสิ้นเปลืองกำลังไฟฟ้าPower Burden	น้อยกว่า 2 W / 10VA
ค่ากระแสเริ่มต้นทำงานStarting Current	20 mA
การทนต่อกระแสเกินOver Current	54A (ต่อเนื่อง 30 นาที)
การทนต่อแรงดันอิมพัลส์Impulse Voltage	6 kV
MECHANICAL CONSTRUCTION	
วัสดุตัวถังBase	วัสดุทนทานป้องกันประเภท 1 (มีขั้วต่อลงดิน)
ระดับการป้องกันน้ำและฝุ่นEnclosure Protection	IP 54 Weather proof (สามารถติดตั้งกลางแจ้งกลางแจ้งฝน)
ขนาดรูของขั้วต่อสายไฟTerminal bore diameter	5.5 mm.
ขนาดของสายไฟApplicable Conductors	10-16 mm ²
น้ำหนักWeight	0.86 kg

4.2 การต่อใช้งานแบบ RS485 Modbus-RTU

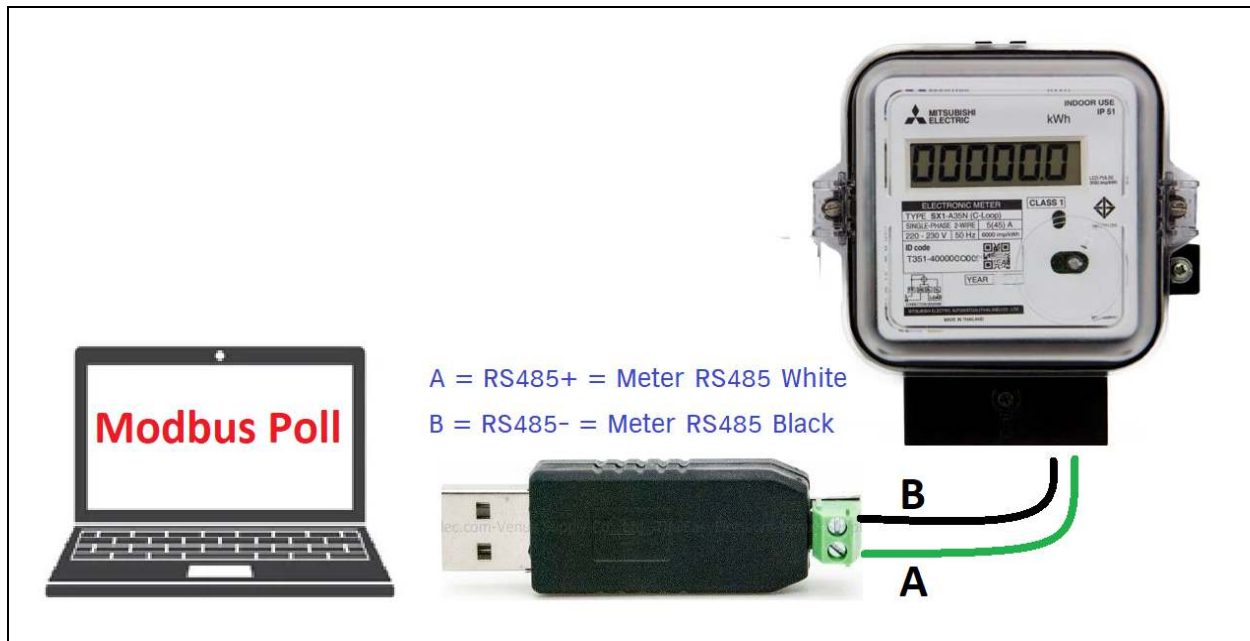


4.3 Communication

Item	Specifications
Physical interface	RS-485 2wires half duplex
Protocol	RTU mode
Transmission wiring type	Multi-point bus (daisy-chain)
Baud rate	1,200 bps.
Data bit	8
Stop bit	1
Parity	Even
CRC polynomial	0xA001
Slave address	1~247 (F7h) (see detail in Appendix B)
Response time	80ms~200ms (programmable) Default 80 ms.
Distance	1,200 m
Max. number	247
Terminator	120 or 150Ω 1/2W
Recommended cable	Shielded twisted pair, recommend LiYCY 2x0.25 mm ²

5. Read SX1-A31E Data with Modbus Poll Software

0/4: Wiring Connection



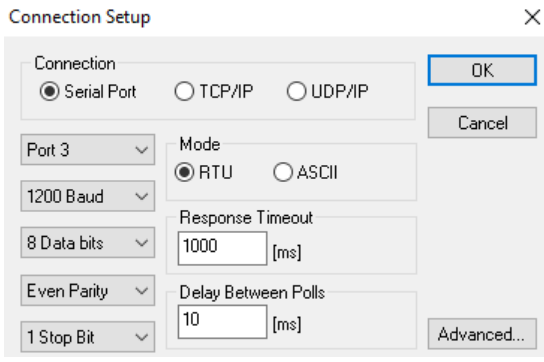
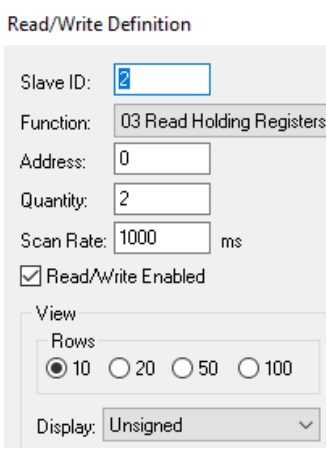
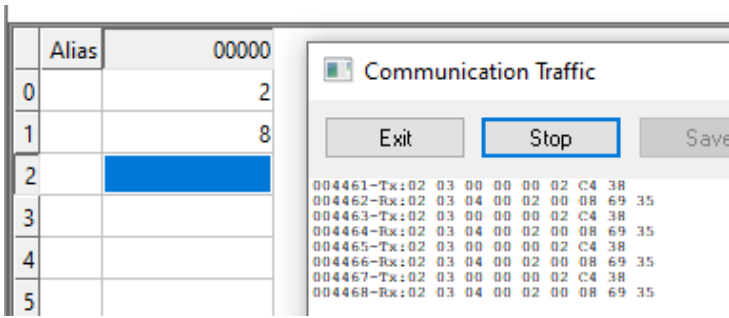
- A = RS485+ = Meter RS485 White
- B = RS485- = Meter RS485 Black

1/4: Read Setup Registers

Register Address		Byte Count	R/W ※1	Register Name	RANGE	Unit
Dec.	Hex.					
40001	0000h	2	R/W	Slave Address ※2 (see detail Appendix B)	1 to 247	-
40002	0001h	2	R/W	Response Time ※3	8 to 20 (default 8)	10ms

Slave Address	Word	Encode	Description	Range	Unit
0x0000 = 0	1	Unsigned	Slave Address	1 to 247	
0x0001 = 1	1	Unsigned	Response Time	8 to 20	10ms

Read with Modbus Poll

 <p>Connection Setup dialog box showing configuration for Serial Port connection. The 'Serial Port' radio button is selected. The 'Port' is set to 'Port 3', 'Baud' is '1200', 'Data bits' is '8', 'Parity' is 'Even', and 'Stop Bit' is '1'. The 'Mode' is set to 'RTU', 'Response Timeout' is '1000' ms, and 'Delay Between Polls' is '10' ms. 'OK', 'Cancel', and 'Advanced...' buttons are visible.</p>	<p>Serial Port Port 3 RTU 1200 8 Bit Event 1 Step</p>
 <p>Read/Write Definition dialog box. 'Slave ID' is '2', 'Function' is '03 Read Holding Registers', 'Address' is '0', 'Quantity' is '2', and 'Scan Rate' is '1000' ms. 'Read/Write Enabled' is checked. Under 'View', 'Rows' are set to '10' and 'Display' is 'Unsigned'.</p>	<p>Slave ID = 2 Function 03 Start Address = 0 Number of word = 2</p> <p>Display = Unsigned</p>
 <p>Main window showing a table with columns 'Alias' and 'Value'. The table has 6 rows, with the second row (index 1) highlighted in blue. The 'Value' column shows '00000' for index 0 and '2' for index 1. A 'Communication Traffic' window is overlaid, showing a list of transactions (Tx/Rx) with addresses and times. The 'Stop' button is highlighted.</p>	<p>Address = 2 Response Time = 80 ms</p>

2/4: Read Instantaneous Value

Register Address		Byte Count	R/W ※1	Register Name	RANGE	Unit
Dec.	Hex.					
40103	0066h	2	R	Line Voltage (RMS)	0 to 65535	0.01V
40106	0069h	2	R	Frequency	0 to 65535	0.1Hz
40113	0070h	2	R	Line Current (RMS)	0 to 65535	0.01A
40116	0073h	2	R	Active Power (W)	0 to 65535	W

Slave Address	Word	Encode	Description	Range	Unit
0x0066 = 102	1	Unsigned	Line Voltage (RMS)	0 to 65535	0.01 V
0x0069 = 105	1	Unsigned	Frequency	0 to 65535	0.1 Hz
0x0070 = 112	1	Unsigned	Line Current (RMS)	0 to 65535	0.01 A
0x0073 = 115	1	Unsigned	Active Power (W)	0 to 65535	W

Read with Modbus Poll

Connection Setup

Connection
☒ Serial Port ☐ TCP/IP ☐ UDP/IP

Port 3 Mode
☒ RTU ☐ ASCII

1200 Baud Response Timeout
1000 [ms]

8 Data bits Delay Between Polls
10 [ms]

Even Parity 1 Stop Bit

OK Cancel Advanced...

Read/Write Definition

Slave ID: 2

Function: 03 Read Holding Registers

Address: 100

Quantity: 16

Scan Rate: 1000 ms

☒ Read/Write Enabled

View
Rows
☒ 10 ☐ 20 ☐ 50 ☐ 100

Display: Unsigned

↑ Read Holding(03H) from 100, 16 Word

	Alias	00100	Alias	00110
0		91		0
1		42898		2445
2		23119		17
3	Volt RMS x 0.01	0	Amp RMS x 0.01	1325
4		17		257
5		500		34
6	Frequency x 0.1 Hz	0	Active Power (W)	
7		2445		
8		0		
9		0		

3/4: Read Counting of Energy Registers

Register Address		Byte Count	R/W ※1	Register Name	RANGE	Unit
Dec.	Hex.					
40111	006Eh	4	R	Active Energy (Wh) imp+exp	0 to 999999999	Wh

Slave Address	Word	Encode	Description	Range	Unit
0x6E = 110	2	Unsigned	Serial Number	0 to 999999999	Wh

Connection Setup

Connection
☒ Serial Port ☐ TCP/IP ☐ UDP/IP

Port 3 Mode
☒ RTU ☐ ASCII

1200 Baud Response Timeout
1000 [ms]

8 Data bits Delay Between Polls
10 [ms]

Even Parity 1 Stop Bit

OK Cancel Advanced...

Slave ID: 2

Function: 03 Read Holding Registers

Address: 110

Quantity: 2

Scan Rate: 1000 ms

☒ Read/Write Enabled

View
Rows
☒ 10 ☐ 20 ☐ 50 ☐ 100

Display: Unsigned

↑ Read Holding(03H) from 110, 2 Word

Alias	00110
0	0
1	2436
2	
3	
4	

Communication Traffic

Exit Stop Save

```

004043-Tx:02 03 00 6E 00 02 A5 E5
004044-Rx:02 03 04 00 00 09 84 CF 00
004045-Tx:02 03 00 6E 00 02 A5 E5
004046-Rx:02 03 04 00 00 09 84 CF 00
004047-Tx:02 03 00 6E 00 02 A5 E5
004048-Rx:02 03 04 00 00 09 84 CF 00
004049-Tx:02 03 00 6E 00 02 A5 E5

```

←

00 00 09 84 = 2436 Wh

≡ Programmer

984

HEX 984

DEC 2,436

4/4: Read General information

Register Address		Byte Count	R/W ×1	Register Name	RANGE	Unit
Dec.	Hex.					
40101	0064h	4	R	Serial No. (see Appendix A)	0 to 9999999	-
40114	0071h	2	R	Current Rating (see definition of reading value Appendix C)	0 to 65535	-

Appendix C Current Rating Register Value Definition**Current Rating register** (register address 0071h)

There are 2 bytes length, MS byte is Basic current and LS byte is Maximum current in ampere.

See table below:

Reading Value (Hex)	Current Rating Definition
05 2D	5(45)A

Diagram showing the mapping of the reading value 05 2D to the current rating definition 5(45)A. The MS byte (05) is labeled as Basic current, and the LS byte (2D) is labeled as Maximum current.

Slave Address	Word	Encode	Description	Range	Unit
0x64 = 100	2	Unsigned	Serial Number	0 to 9999999	
0x71 = 113	1	Unsigned	Current Rating	0 to 65535	

Read/Write Definition

Slave ID:

Function:

Address:

Quantity:

Scan Rate: ms

☒ Read/Write Enabled

View

Rows: ☒ 10 ☐ 20 ☐ 50 ☐ 100

Display:

Read/Write Definition

Slave ID:

Function:

Address:

Quantity:

Scan Rate: ms

☒ Read/Write Enabled

View

Rows: ☒ 10 ☐ 20 ☐ 50 ☐ 100

Display:

Alias	Value
0	91
1	42898
2	005B A792
3	
4	

Communication Traffic

Exit Stop Save

```

005377-Tx:02 03 00 64 00 02 85 E7
005378-Rx:02 03 04 00 5B A7 92 43 7D
005379-Tx:02 03 00 64 00 02 85 E7
005380-Rx:02 03 04 00 5B A7 92 43 7D
005381-Tx:02 03 00 64 00 02 85 E7
005382-Rx:02 03 04 00 5B A7 92 43 7D
005383-Tx:02 03 00 64 00 02 85 E7

```

←

005BA792 = 6006674

S/N = 6006674

5B A792

HEX 5B A792

DEC 6,006,674

Alias	Value
0	05 2D
1	05 = 5(A)
2	2D = 45(A)
3	Rate = 5(45)A

Communication Traffic

Exit Stop

```

005975-Tx:02 03 00 71 00 01 D4 22
005976-Rx:02 03 02 05 2D 3F 09
005977-Tx:02 03 00 71 00 01 D4 22
005978-Rx:02 03 02 05 2D 3F 09
005979-Tx:02 03 00 71 00 01 D4 22
005980-Rx:02 03 02 05 2D 3F 09

```

←

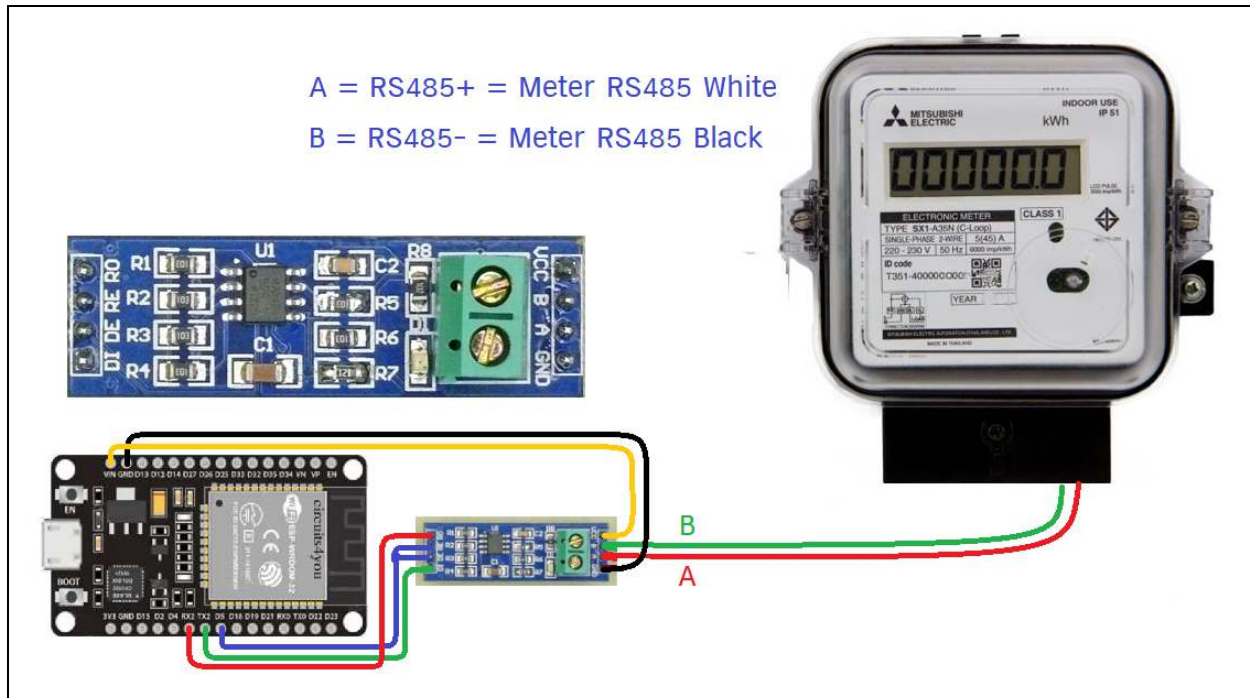
05 2D

Operate = 0x05 = 5(A)

Max = 0x2D = 45(A)

6 Read SX1-A31E Data with Arduino ESP-32

6.1 Wiring

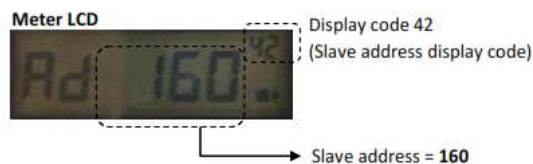


6.2 Modbus Slave ID

Appendix B Slave Address

A slave address of any meter is shown on meter LCD by auto scrolling display. The slave address shown by display item, code 42.

<Example B1>



Note: Meter which not show Slave Address display item (Ad) does not support Modbus protocol.
Please contact factory.

A slave address must be unique on a Modbus serial bus. If some slave addresses are duplicated on bus, slave address changing must be done by software "Modbus Meter Setting" (download setting software from our website: www.meath-co.com/meter)

6.3 Add **ModbusMaster by Doc Walker V2.0.1** and test this code**ModbusMaster**

by Doc Walker 4-20ma@wvfans.net>

Enlighten your Arduino to be a Modbus master. Enables communication with Modbus slaves over RS232/485 (via RTU protocol). Requires an RS232/485 transceiver.

[More info](#)

Version 2.0.1

Install

6.4 Test This Code

```

#include <ModbusMaster.h>
#define RX_PIN    16 // Rx Pin
#define TX_PIN    17 // Tx Pin
#define RS485Control  5 // RS485 Direction control
#define Pin_LEDMonitor 2

#define Slave_ID 2
ModbusMaster modbus;

void preTransmission() {
  digitalWrite(RS485Control, HIGH);
}

void postTransmission() {
  digitalWrite(RS485Control, LOW);
}

void setup() {
  pinMode(RS485Control, OUTPUT);
  digitalWrite(RS485Control, LOW);
  Serial.begin(115200, SERIAL_8N1);
  Serial2.begin(1200, SERIAL_8E1, RX_PIN, TX_PIN);
  modbus.begin(Slave_ID, Serial2);
  modbus.preTransmission(preTransmission);
  modbus.postTransmission(postTransmission);
}

void loop() {
  uint8_t result;
  uint16_t data[6];
  Serial.println();

  // Read 1 Registers starting Addr.102(Volt)
  result = modbus.readHoldingRegisters(102, 1);
  if (result == modbus.ku8MBSuccess) {
    float VLINE = modbus.getResponseBuffer(0x00) / 100.0;
    Serial.print("\tLINE RMS.Voltage(V): ");
    Serial.print(VLINE);
  }
  delay(500);

  // Read 2 Registers starting Addr.110(Active Power)
  result = modbus.readHoldingRegisters(110, 2);
  if (result == modbus.ku8MBSuccess) {
    int HPower = modbus.getResponseBuffer(0x00);
    int LPower = modbus.getResponseBuffer(0x01);
    float APower = ((HPower<<16)|LPower)/1000.0;
    Serial.print("\tLINE Active Power(kWh): ");
    Serial.print(APower);
  }
  delay(500);
}

```

```

LINE RMS.Voltage(V): 223.27      LINE Active Power(kWh): 2.49
LINE RMS.Voltage(V): 224.65      LINE Active Power(kWh): 2.49
LINE RMS.Voltage(V): 224.96      LINE Active Power(kWh): 2.49
LINE RMS.Voltage(V): 223.12      LINE Active Power(kWh): 2.49
LINE RMS.Voltage(V): 222.81      LINE Active Power(kWh): 2.49
LINE RMS.Voltage(V): 222.99      LINE Active Power(kWh): 2.49
LINE RMS.Voltage(V): 223.12      LINE Active Power(kWh): 2.49
LINE RMS.Voltage(V): 223.12      LINE Active Power(kWh): 2.49
LINE RMS.Voltage(V): 224.37

```

☐ Show timestamp

Newline

115200 baud

7. Send Data to ThingsBoard

7.1 Add 3 Library

- ThingsBoard by ThingsBoard Team Ver-0.5.0
- PubSubClient by Nick O'Leary Ver-2.8.0
- ArduinoJson by Benoit Blanchon Ver-6.18.1

The screenshot shows the Arduino IDE library manager interface with three libraries listed:

- ThingsBoard** by ThingsBoard Team: ThingsBoard library for Arduino. A library for connecting to the ThingsBoard IoT platform. Thin wrapper on PubSubClient. Version 0.5.0. [More info](#). [Install](#)
- PubSubClient** by Nick O'Leary: A client library for MQTT messaging. MQTT is a lightweight messaging protocol ideal for small devices. This library allows you to send and receive MQTT messages. It supports the latest MQTT 3.1.1 protocol and can be configured to use the older MQTT 3.1 if needed. It supports all Arduino Ethernet Client compatible hardware, including the Intel Galileo/Edison, ESP8266 and TI CC3000. Version 2.8.0. [More info](#). [Install](#)
- ArduinoJson** by Benoit Blanchon: A simple and efficient JSON library for embedded C++. ArduinoJson supports ✓ serialization, ✓ deserialization, ✓ MessagePack, ✓ fixed allocation, ✓ zero-copy, ✓ streams, ✓ filtering, and more. It is the most popular Arduino library on GitHub ♥♥♥♥♥. Check out arduinojson.org for a comprehensive documentation. Version 6.18.3. [More info](#). [Install](#)

7.2 Create Device on ThingsBoard and Copy Token ID

The screenshot shows the ThingsBoard web interface for a device named **TB_D40_RxData**. The "Device details" tab is selected, showing a navigation bar with tabs: Details, Attributes, Latest telemetry, Alarms, and Events. Below the tabs are four action buttons: "Make device public", "Assign to customer", "Manage credentials", and "Delete". At the bottom, there are two buttons: "Copy device Id" and "Copy access token". A mouse cursor is hovering over the "Copy access token" button. A hash symbol (#) is visible at the bottom right of the interface.

7.2 Test This Code

```

// ThingsBoard by ThingsBoard Team Ver-0.5.0
// PubSubClient by Nick O'Leary Ver-2.8.0
// ArduinoJson by Benoit Blanchon Ver-6.18.1

#include "ThingsBoard.h"
#include <WiFi.h>

#define Wifi_Name    "Test1234"
#define Wifi_Password "0816601929"
#define ThingsBoard_Token "qr11iKBccKa0CLpLjUyV"
#define ThingsBoard_Server "demo.thingsboard.io"

#define SERIAL_DEBUG_BAUD 115200

WiFiClient espClient;
ThingsBoard tb(espClient);

int status = WL_IDLE_STATUS;
float LNVolt, APower = 0;
char latitude[] = "14.785937521295068";
char longitude[] = "102.02638983327009";
char stsName[] = "Rooks Korat Country Club";

void setup() {
  Serial.begin(SERIAL_DEBUG_BAUD);
  WiFi.begin(Wifi_Name, Wifi_Password);
  InitWiFi();
}

void loop() {
  delay(5000);

  if (WiFi.status() != WL_CONNECTED) {
    reconnect();
  }

  if (!tb.connected()) {
    Serial.print("Connecting to: ");
    Serial.print(ThingsBoard_Server);
    Serial.print(" with token ");
    Serial.println(ThingsBoard_Token);
    if (!tb.connect(ThingsBoard_Server, ThingsBoard_Token)) {
      Serial.println("Failed to connect");
      return;
    }
  }

  Serial.println("Sending data...");
  LNVolt = 210 + random(100, 900) / 100.0;
  APower = APower + random(10, 90) / 100.0;
  tb.sendTelemetryString("stsName", stsName);
  tb.sendTelemetryString("latitude", latitude);
  tb.sendTelemetryString("longitude", longitude);
  tb.sendTelemetryFloat("Lne_Voltage", LNVolt);
  tb.sendTelemetryFloat("Abs_Power", APower);
  tb.loop();
}

void InitWiFi() {
  Serial.println("Connecting to AP ...");
  WiFi.begin(Wifi_Name, Wifi_Password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("Connected to AP");
}

void reconnect() {
  status = WiFi.status();
  if (status != WL_CONNECTED) {
    WiFi.begin(Wifi_Name, Wifi_Password);
    while (WiFi.status() != WL_CONNECTED) {
      delay(500);
      Serial.print(".");
    }
    Serial.println("Connected to AP");
  }
}

```

การใช้งาน ThingsBoard IoTs Platform เพื่อสร้างและจัดการระบบอัจฉริยะ
ThingsBoard IoTs Platform for smart system

ชื่อ-สกุล :

5/5 -- คำถามท้ายบทเพื่อทดสอบความเข้าใจ

Quiz_401 – ทดสอบการใช้งาน Rule Chain เพื่อแจ้งเตือนไปยัง LINE (ตาม Lab-401)

- ทำการทดสอบตามเอกสาร Lab-401

รูปการทดสอบ 1 – Rule Chain
รูปการทดสอบ 2 – Dashboard
รูปการทดสอบ 3
รูปการทดสอบ 4
อธิบายแนวทางการปรับใช้กับงานที่ตัวเองรับผิดชอบ

Quiz_402 – ทดสอบการทำงาน Alarm เมื่ออุณหภูมิอยู่นอกเขตที่กำหนด (ตาม Lab-402)

- ทำการทดสอบตามเอกสาร Lab-402 กำหนดเงื่อนไขในช่วงที่ยอมรับ คือ temperature = [-5,15] และ humidity = [40 – 60]%

รูปการทดสอบ 1 – Rule Chain
รูปการทดสอบ 2 – Dashboard
รูปการทดสอบ 3
รูปการทดสอบ 4
อธิบายแนวทางการปรับใช้กับงานที่ตัวเองรับผิดชอบ

Quiz_403 – ให้ตอบคำถาม แสดงแนวคิด อภิปรายในหัวข้อต่อไปนี้

1. ความรู้ที่ได้เพิ่มเติมเกี่ยวกับ IoT

2. ความรู้ที่ได้เพิ่มเติมเกี่ยวกับ ThingsBoard

3. แนวทางการปรับใช้ ThingsBoard IoT Platform กับงานที่รับผิดชอบ

4. คำแนะนำ ข้อเสนอแนะ จากผู้เรียน – ประเด็นเนื้อหาที่นำเสนอ (มากไป, น้อยไป, ลึกไป, อธิบายน้อยไป, เอกสาร, ความเหมาะสมของเวลา)

5. คำแนะนำ ข้อเสนอแนะ จากผู้เรียน – ประเด็นเนื้อหาที่อยากให้เสริม หรือเปิดหลักสูตรเพิ่มเติม หรือต้องการให้อบรมแบบเข้าห้องเรียน
