Introducing NETPIE2020 & Workshop

Mr. Piyawat Jomsathan (Tae)
Cyber-Physical Systems (CPS)
National Electronics and Computer Technology Center (NECTEC), Thailand

Note : This powerpoint is modified significantly by Varodom Toochinda to keep up with the latest version of NETPIE 2020.
Some slides may still show user interface from previous version when changes are deemed unnecessary.

Personal Contact

NETPIE2020 Trainer

Cyber Physical Systems (CPS)

NECTEC Building Room 418

Mr.Piyawat Jomsathan

Nickname : Tae

Age : 23

E-mail : piyawat.jom@nectec.or.th

Tel. 02-564-6900 ext. 2469

Mr.Natapon Tansangworn

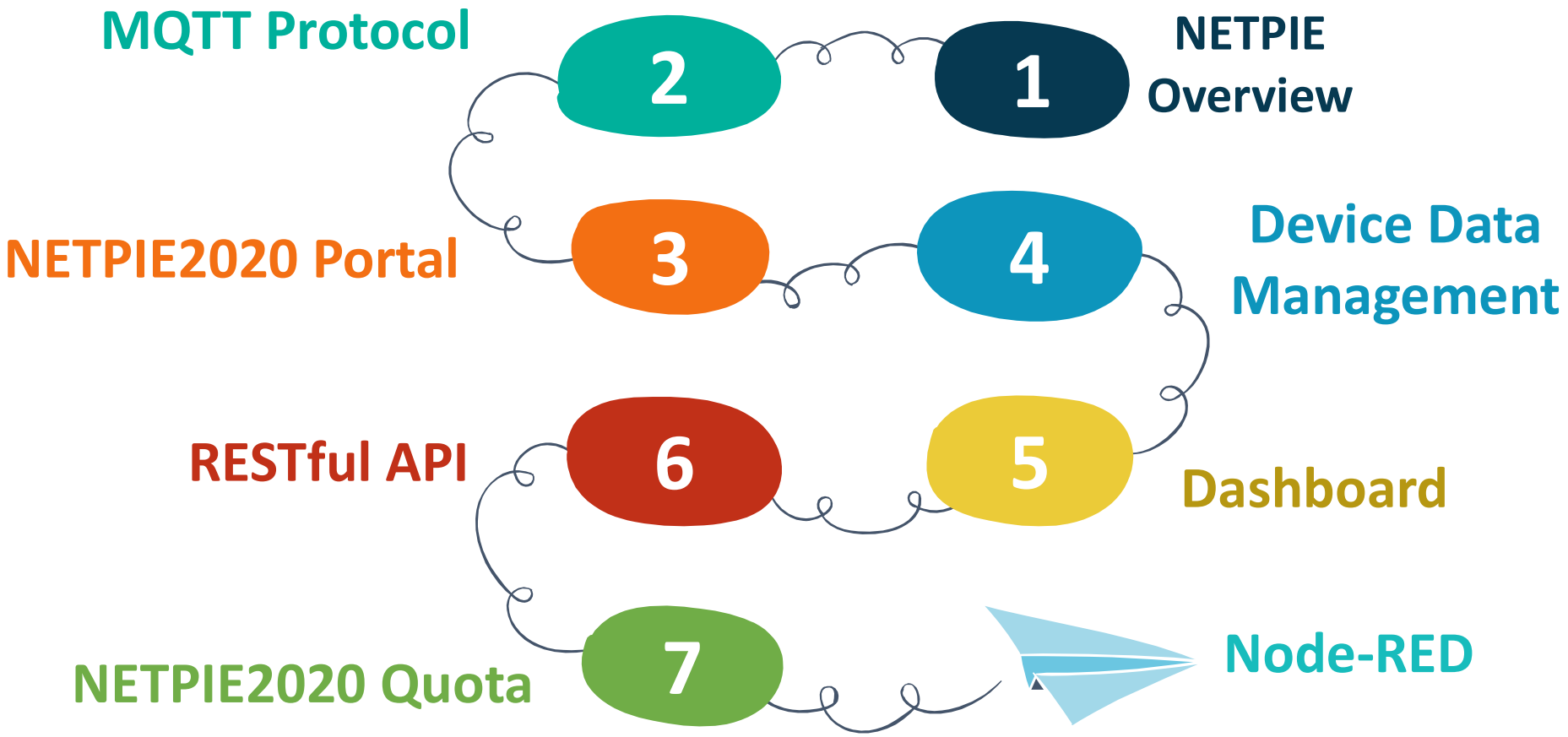
Nickname : Prig

Age : 28

E-mail : natapon.tan@nectec.or.th

Tel. 02-564-6900 ext. 2463

Schedule


MQTT Protocol
2
1
NETPIE
Overview
NETPIE2020 Portal
3
4
Device Data
Management
RESTful API
6
5
Dashboard
NETPIE2020 Quota
7
Node-RED

NETPIE Overview

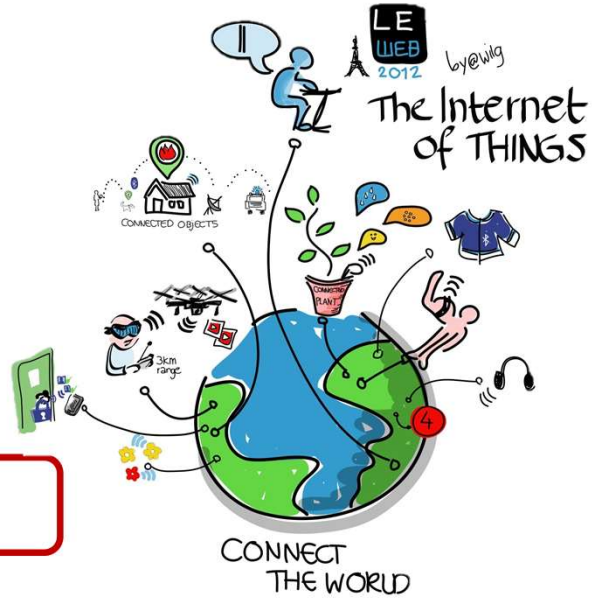
Module 1 : IoT & NETPIE

What is NETPIE?

NETPIE is an IoT cloud platform that is open to the public. The platform will help devices able to communicate with each other, receive - transmit data between devices in real-time, allowing users to know the information of the device at that time, no matter where the user is.

Main features of NETPIE

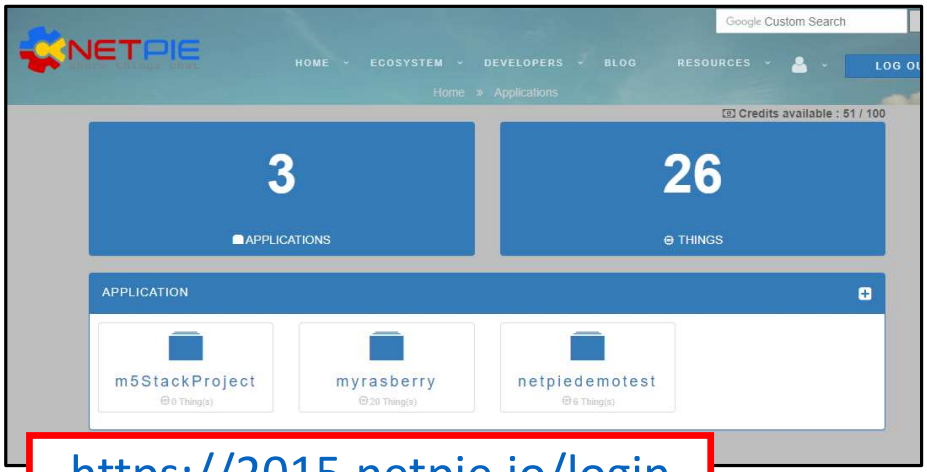
1. Monitoring : to display device or sensor data in real time
2. Controlling : to control the operation of devices via the cloud platform
3. Data Storage : to collect data from sensors or devices
4. Notification : to alert the user when sensor malfunction occurs



Module 1 : IoT & NETPIE

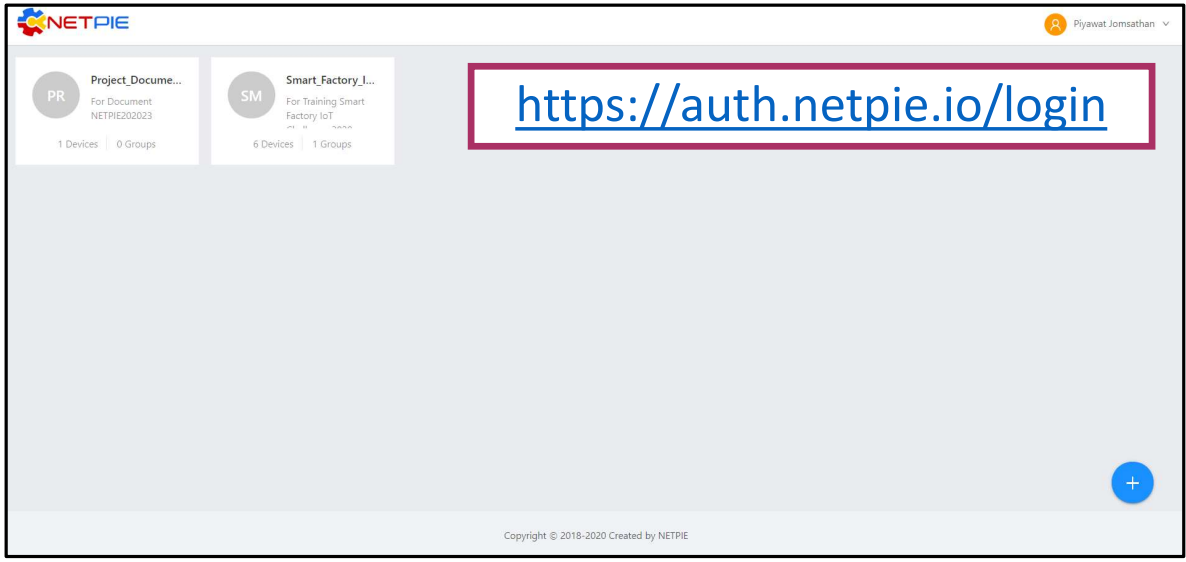
At present, NETPIE is developed in 2 versions
NETPIE 2015 and NETPIE 2020

NETPIE 2015



https://2015.netpie.io/login

NETPIE 2020



https://auth.netpie.io/login

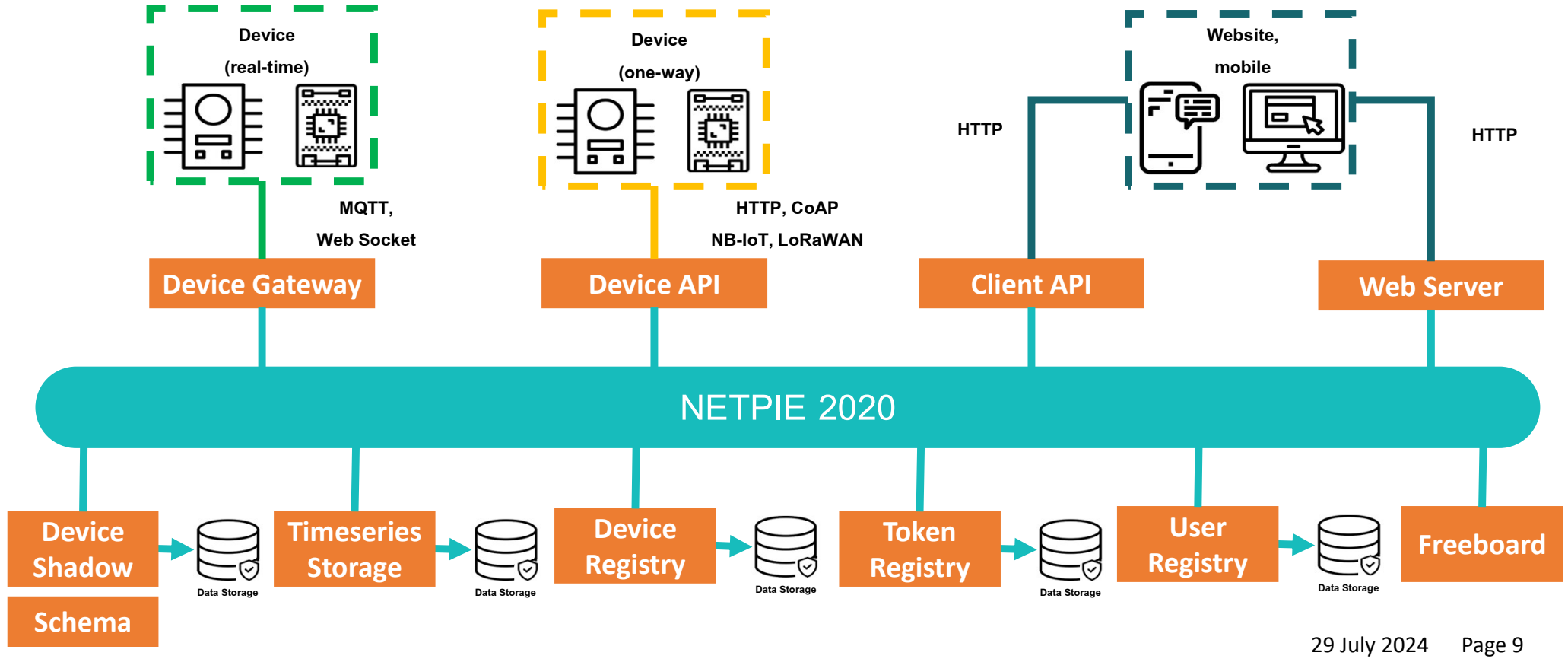
Module 1 : IoT & NETPIE

NETPIE 2020 and NETPIE 2015comparison

	NETPIE 2020	NETPIE 2015
Design Philosophy	Platform - Centric	Device – Centric
Commercial Ability	Commercial – Ready	Need to re-program Hardware migrate to commercial platform
Suitable Usage (1)	Mass production, Project-based	Project-based
Target User	IoT consumer product makers, Hobbyists, Students	Makers, hobbyists, students
Communication Protocol	MQTT, HTTP	Microgear
Programming Language	Any languages with MQTT library support	Limited to Microgear library
Hardware Support (2)	Unlimited as long as it supports MQTT	Limited to those with Microgear support
Device Identity and Group	No APPID Device identity and group can be adjusted after product is sold/installed	Use APPID Device identity and group must be programmed into firmware
Rate Limit	Allow burst	Everyone is subject to the same rate limit
Trigger (3)	Can set trigger action in cloud platform	Set trigger action inside IoT devices

NETPIE Overview

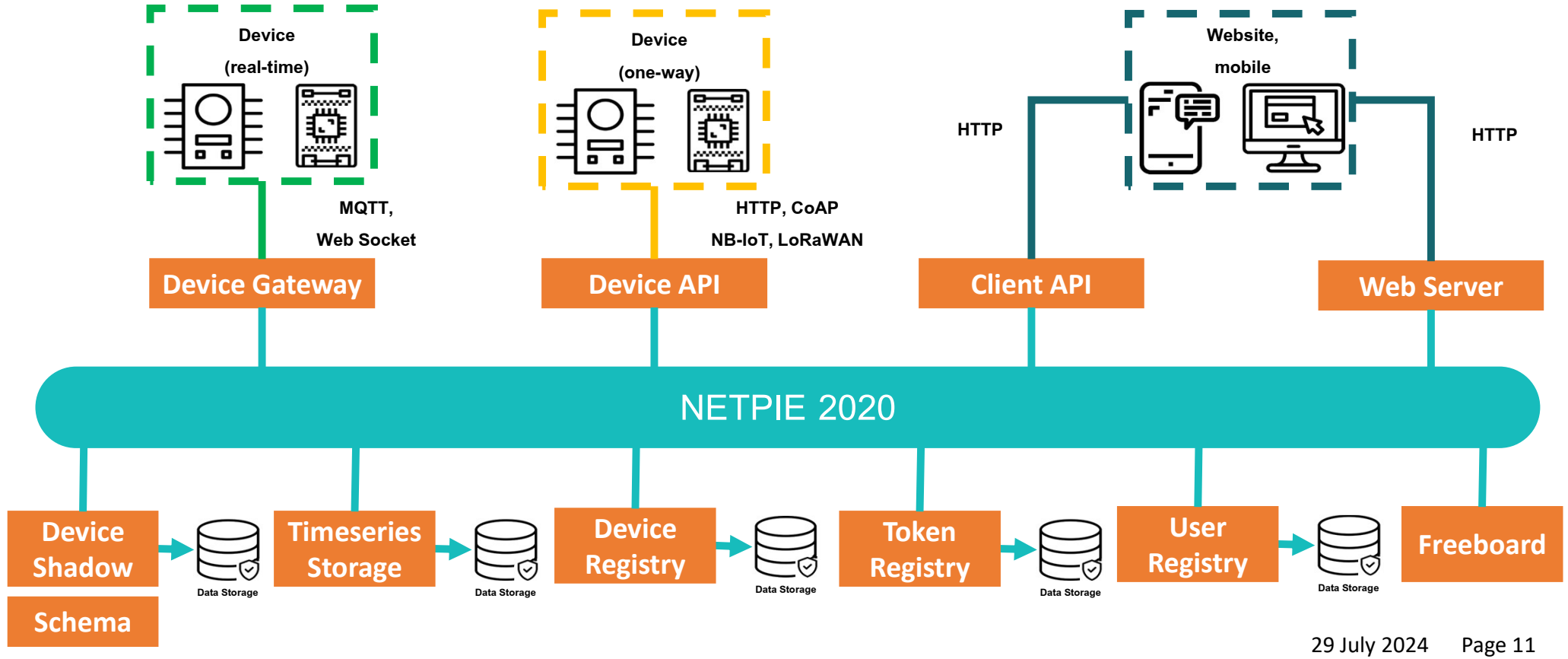
NETPIE 2020 Architecture



MQTT Protocol

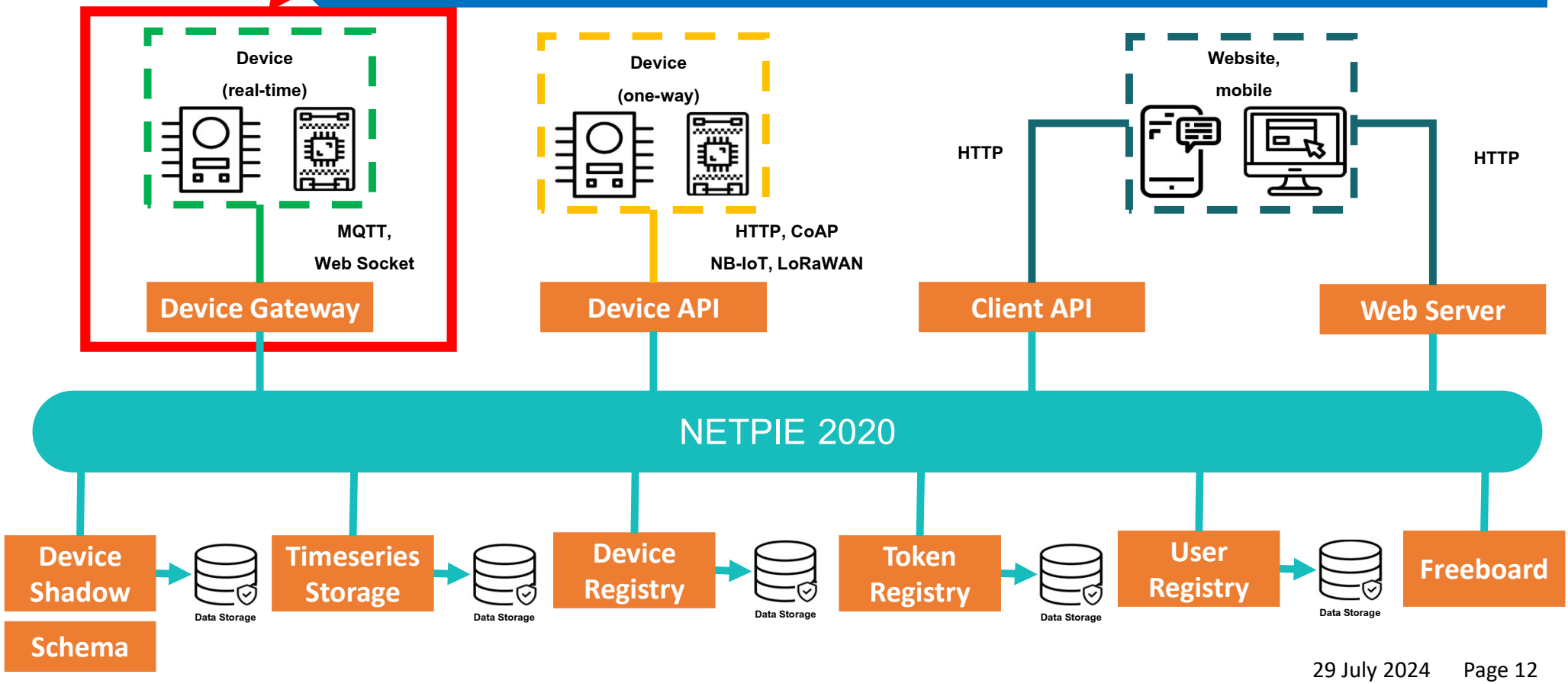
NETPIE Overview

NETPIE 2020 Architecture



MQTT Protocol

If real-time monitoring is required on NETPIE2020, a device gateway is required to communicate with the MQTT protocol.

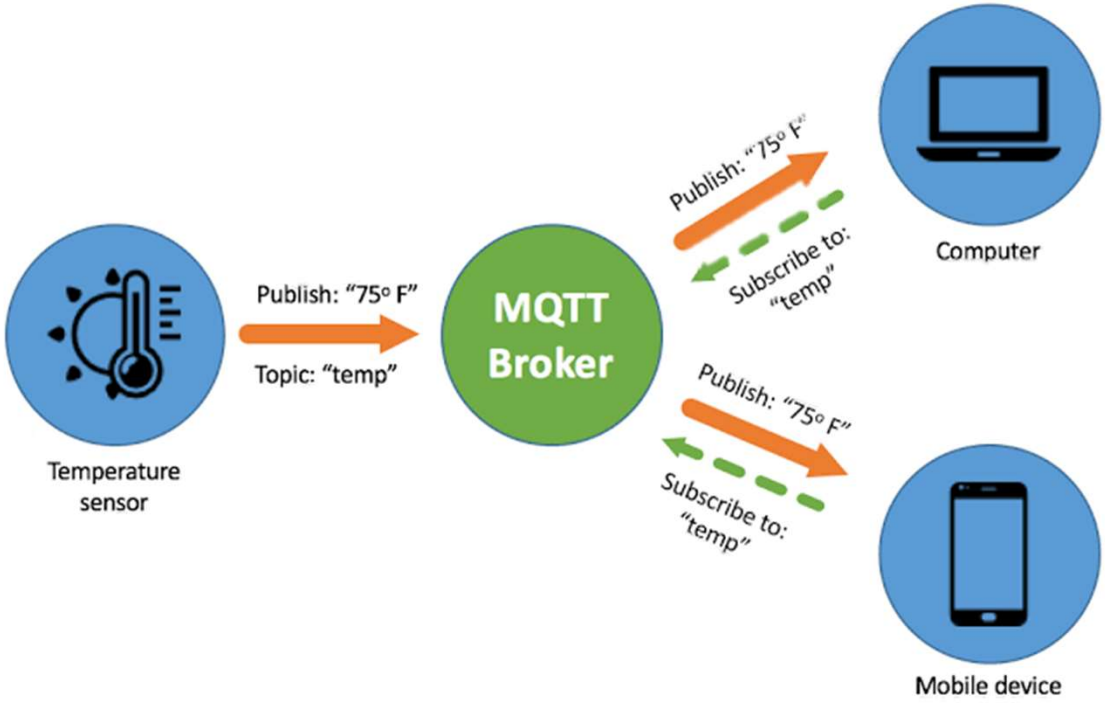


MQTT Protocol

NETPIE2020 communication (MQTT)

What is MQTT ??

- MQTT is a protocol designed to connect M2M or device to device, which is supported by IoT
- Use the principle of data transmission, Publish / Subscribe, similar to the principles used in the Web Service that requires a Web Server as an intermediary between users' computers.
- But MQTT uses an intermediary called Broker that manages the transmission order between devices and Publish / Subscribe mechanisms.



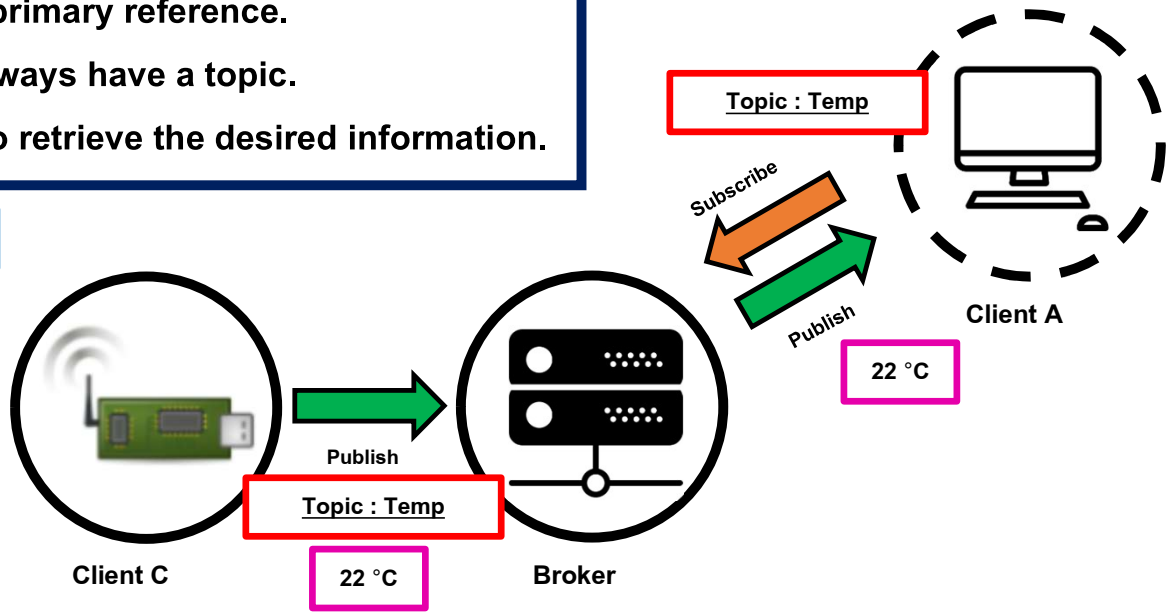
MQTT Protocol

transmissions on the MQTT have a topic as their primary reference.

- **The data to be published to the Broker must always have a topic.**
- **On the subscribe end, it must refer to a topic to retrieve the desired information.**

MQTT communication example

- Client C publishes information on a topic named Temp, while Client A subscribes to topic Temp.
- The data from Client C is a temperature with values equal to 22 ° C, so Client A receives the information.



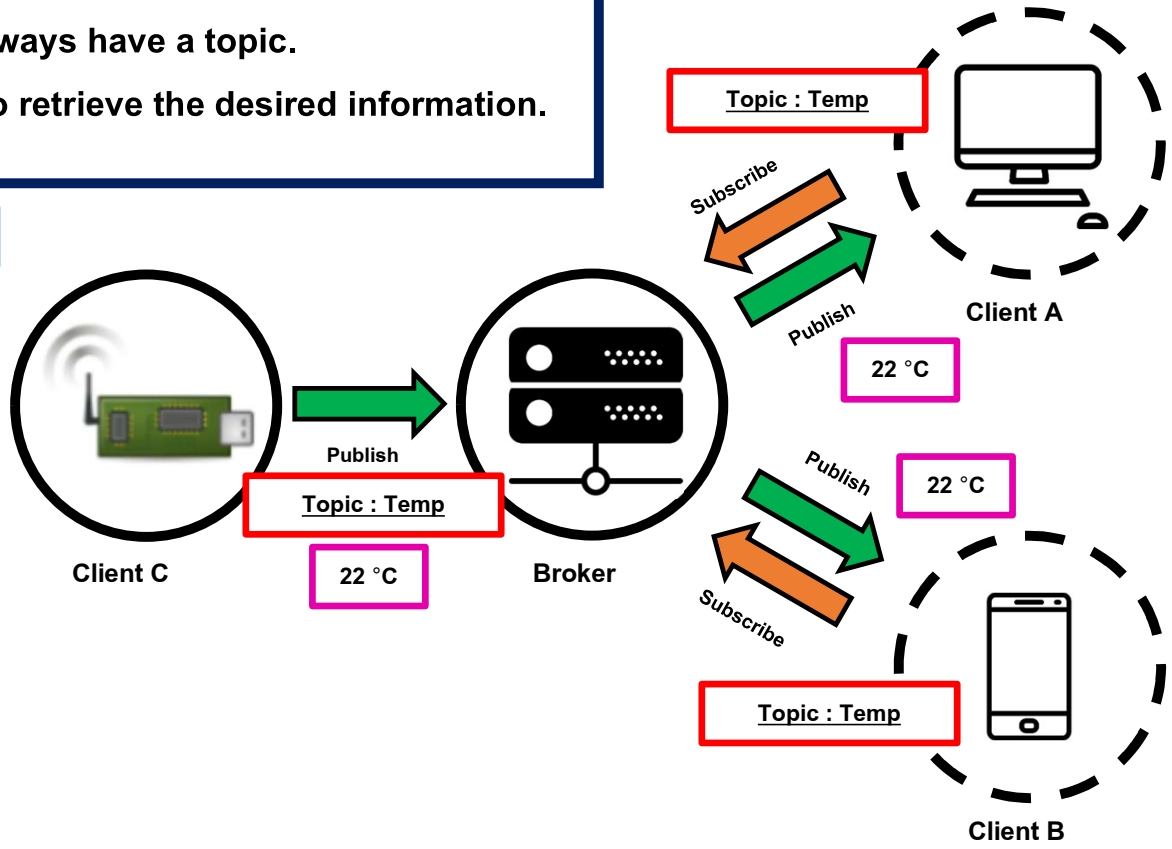
MQTT Protocol

transmissions on the MQTT have a topic as their primary reference.

- **The data to be published to the Broker must always have a topic.**
- **On the subscribe end, it must refer to a topic to retrieve the desired information.**

MQTT communication example

- Client C publishes information on a topic named Temp, while Client A subscribes to topic Temp.
- The data from Client C is a temperature with values equal to 22 ° C, so Client A receives the information.
- Later, Client B subscribes to topic Temp as well. The stored temperature is immediately sent to Client B after subscribing.



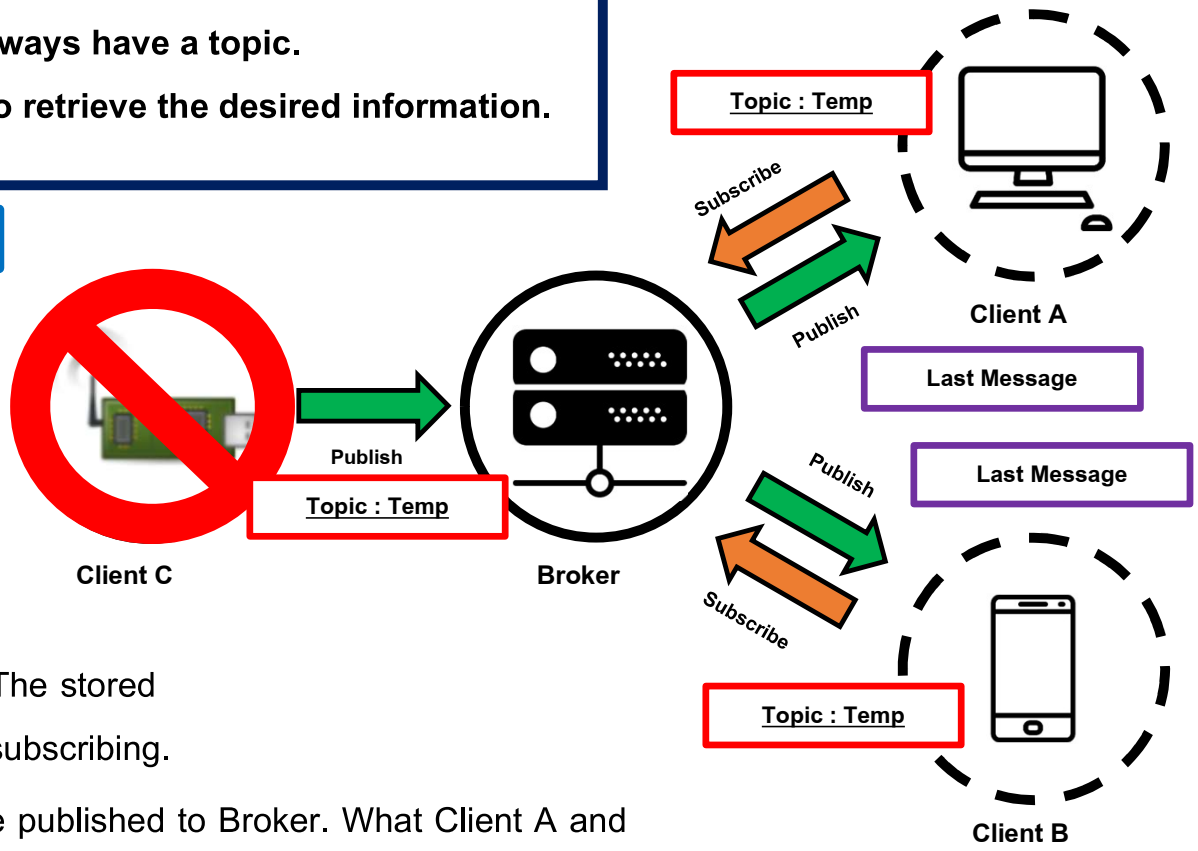
MQTT Protocol

transmissions on the MQTT have a topic as their primary reference.

- **The data to be published to the Broker must always have a topic.**
- **On the subscribe end, it must refer to a topic to retrieve the desired information.**

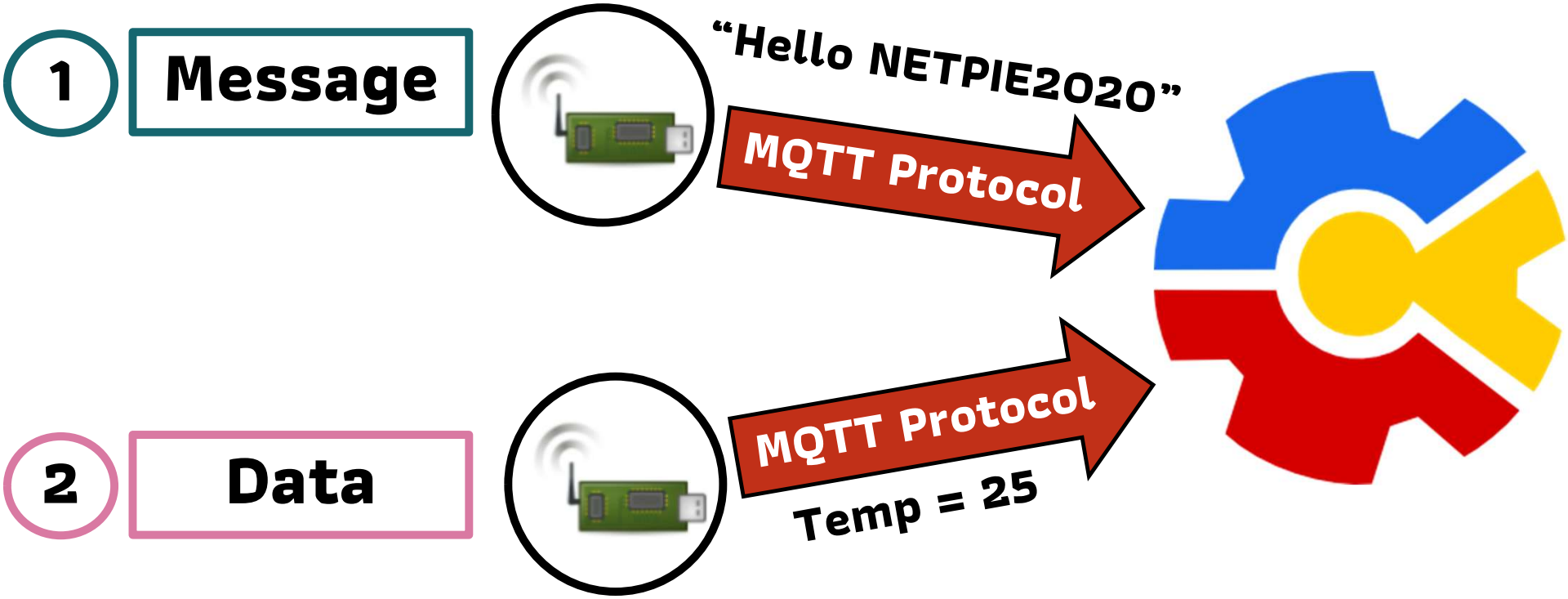
MQTT communication example

- Client C publishes information on a topic named Temp, while Client A subscribes to topic Temp.
- The data from Client C is a temperature with values equal to 22 ° C, so Client A receives the information.
- Later, Client B subscribes to topic Temp as well. The stored temperature is immediately sent to Client B after subscribing.
- But when Client C is disconnected, no data will be published to Broker. What Client A and B display is the last message sent by Client C



MQTT Protocol

There are two types of messages sent to NETPIE2020 via MQTT.



MQTT Protocol

MQTT on NETPIE2020 has two types of transmission

1. **Message**

   Send a message via MQTT by referring to a topic. The format is @msg/topic

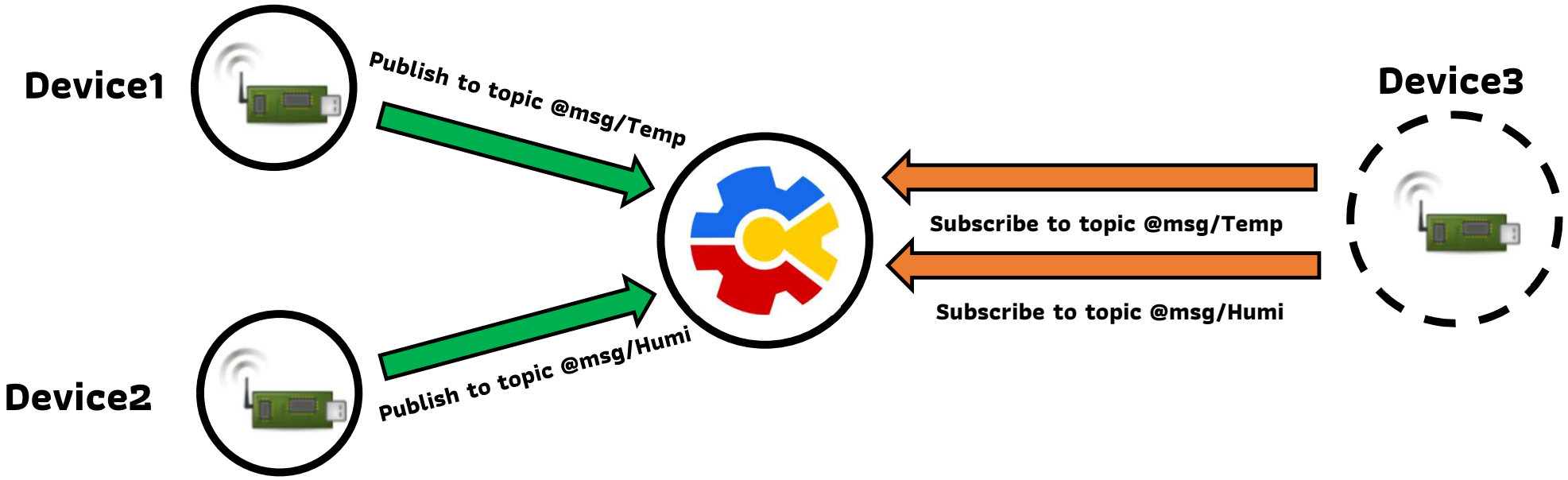
2. **Shadow [Data]**

   Send message via MQTT to Device Shadow, to record latest message. The topic is @shadow/data/update with message in JSON format. For example, {data:{temp:24}}

MQTT Protocol

Wildcard Topic

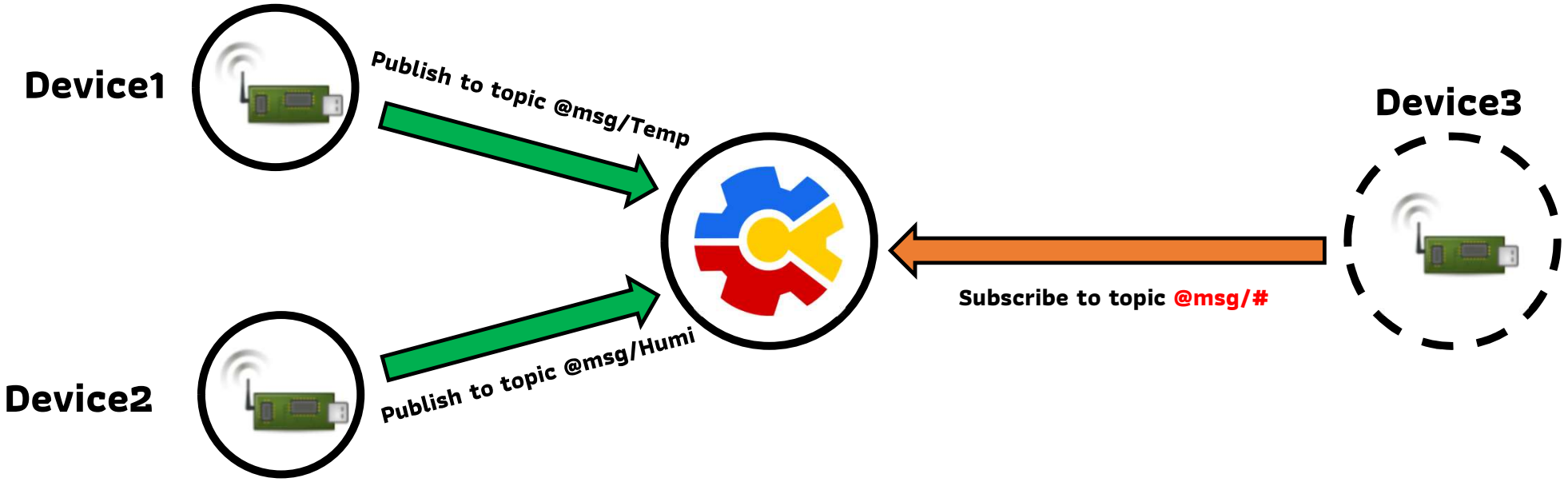
If you want Device 3 to receive messages from Device1 and Device2, you need to subscribe to 2 topics



MQTT Protocol

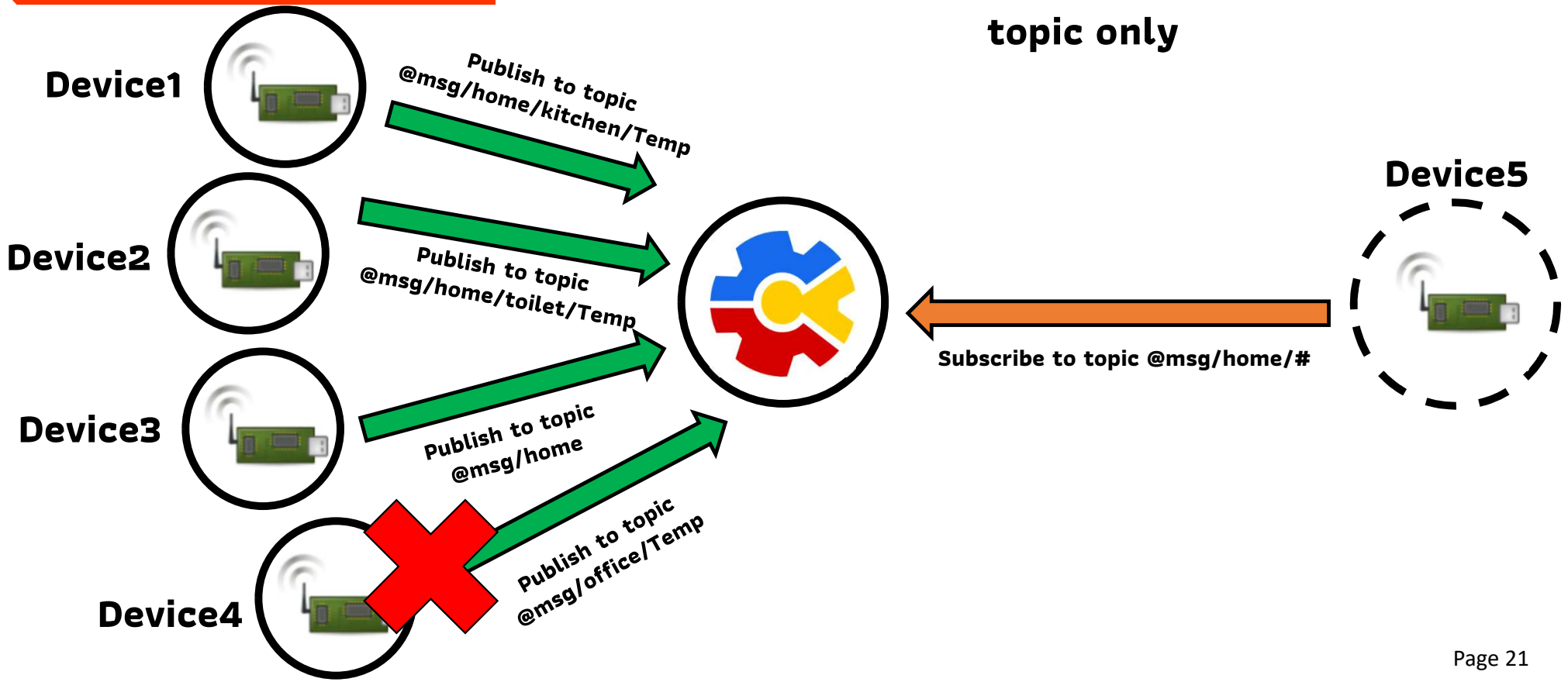
Wildcard Topic

The symbol # in part of a topic represents every word and all section of words within / after that position. If Subscribe topic / # is equivalent to subscribe all.



MQTT Protocol

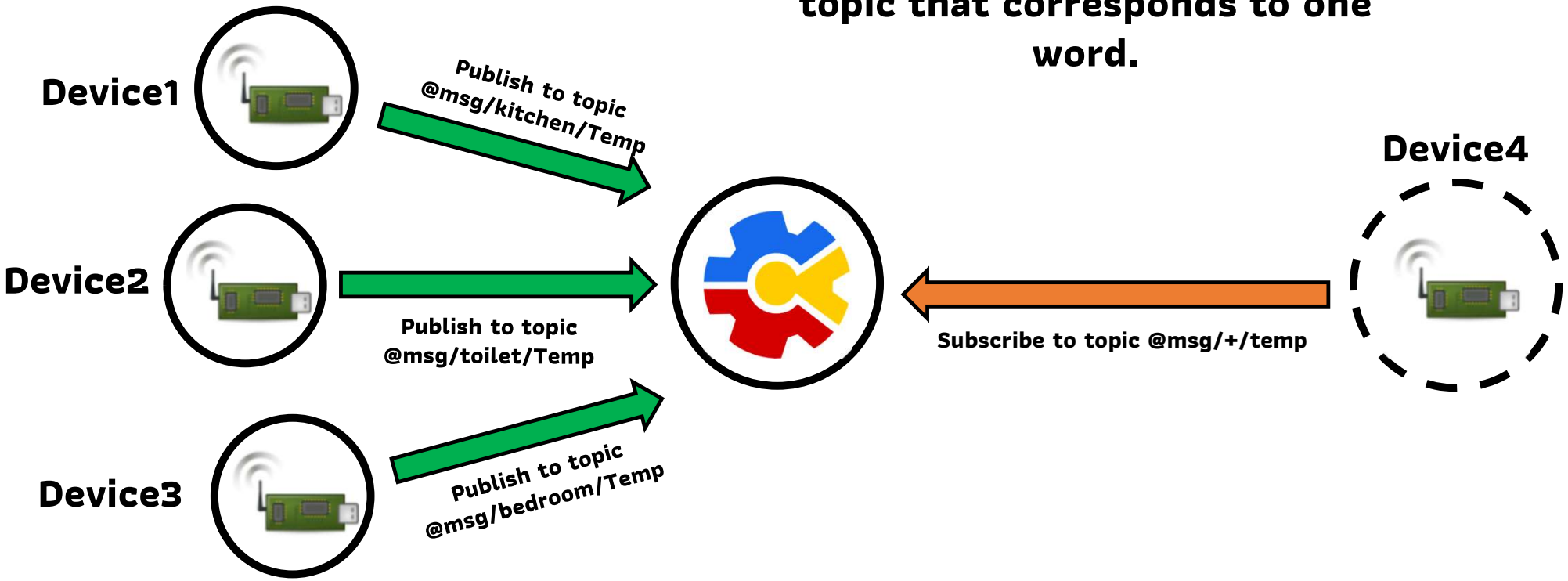
Wildcard Topic



MQTT Protocol

Wildcard Topic

The + symbol can be used in a topic that corresponds to one word.



MQTT Ports

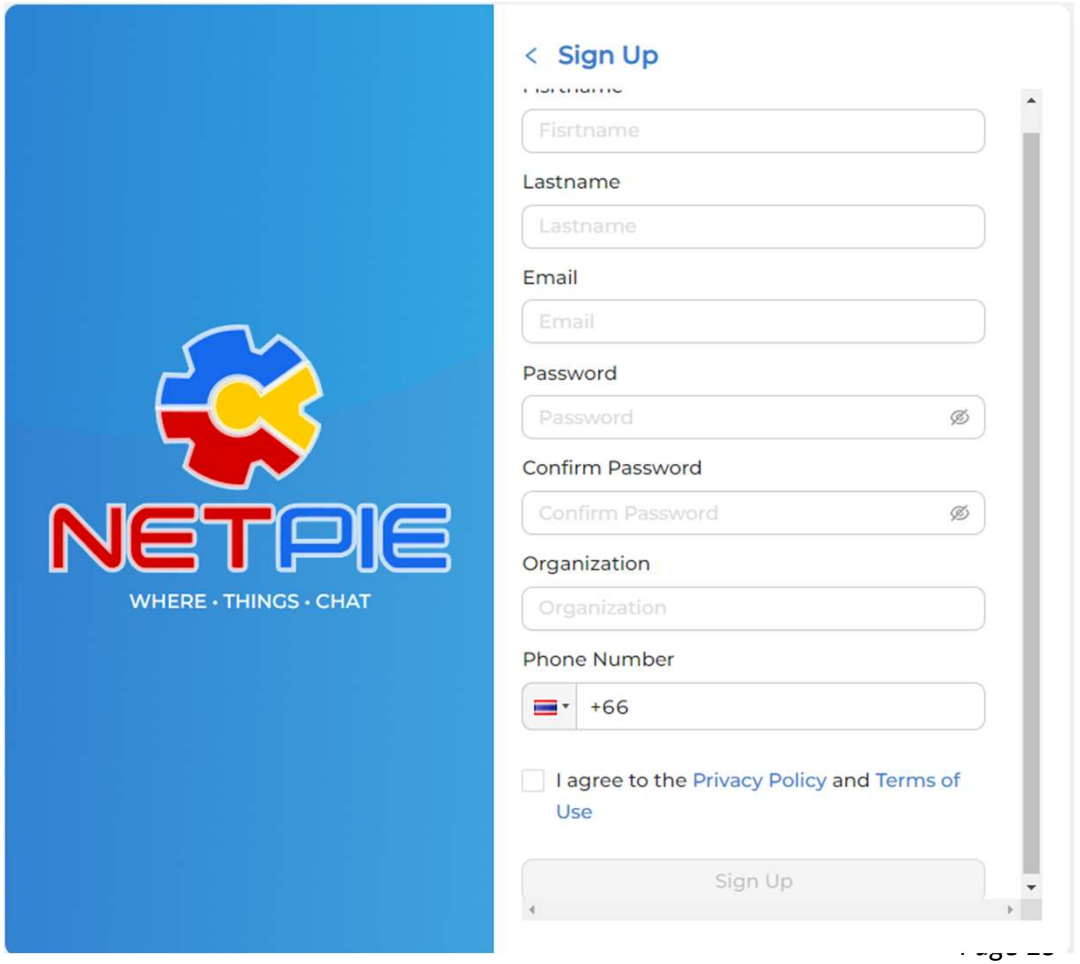
Connection	Port
mqtt/tcp (broker.netpie.io)	1883
mqtts/tls (broker.netpie.io)	8883
ws (mqtt.netpie.io/mqtt)	80
wss (mqtt.netpie.io/mqtt)	443

NETPIE 2020 Portal

NETPIE 2020 Portal

Getting started with NETPIE2020

Create NETPIE2020 user account at
https://authx.netpie.io/signup

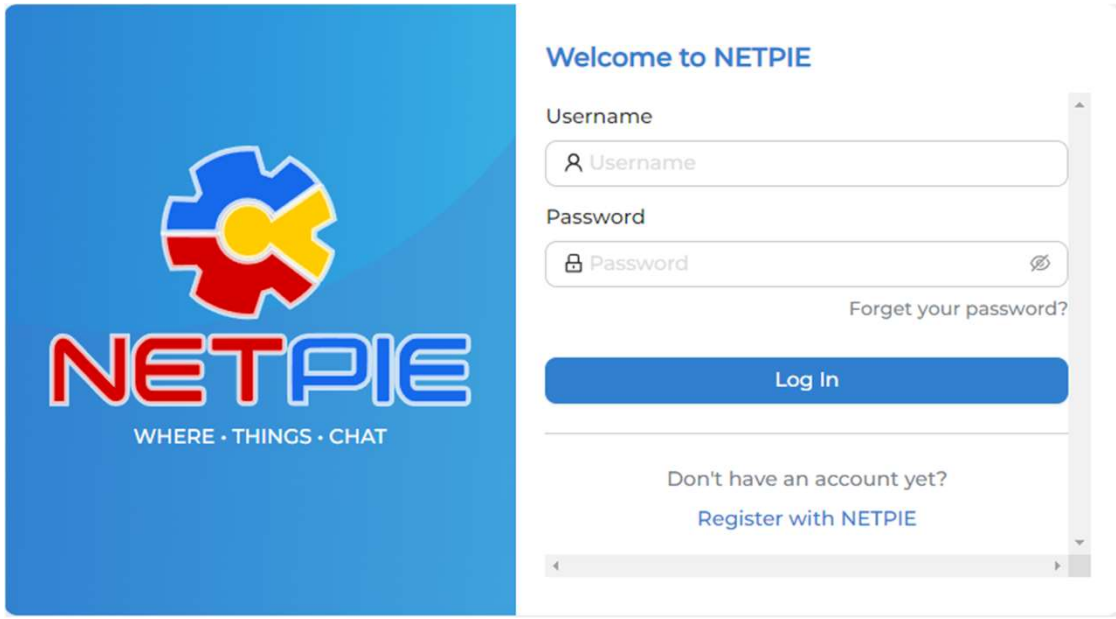


NETPIE 2020 Portal

Getting started with NETPIE2020

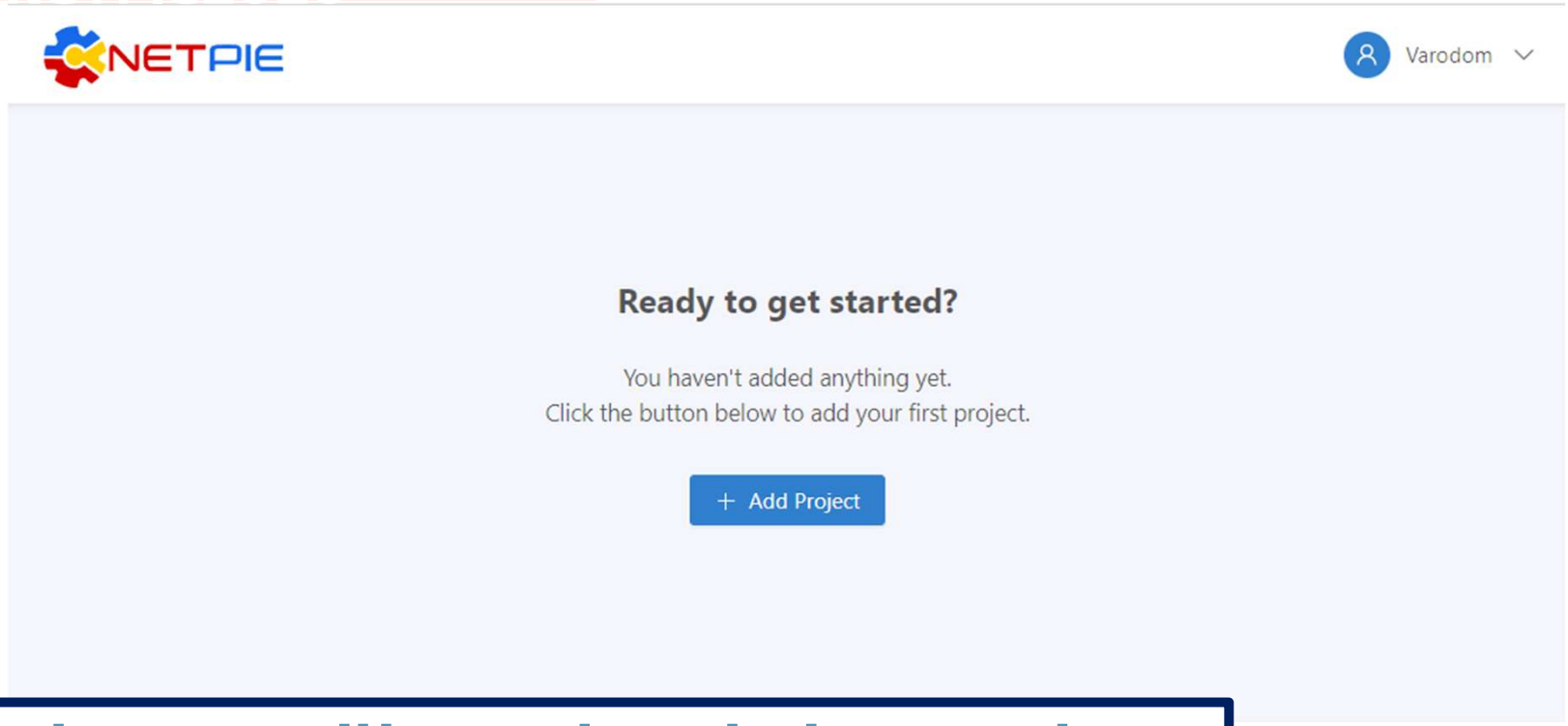
After sign up NETPIE2020, login With Username (email) and Password

Or login at this link
https://authx.netpie.io/login



NETPIE 2020 Portal

Getting started with NETPIE2020

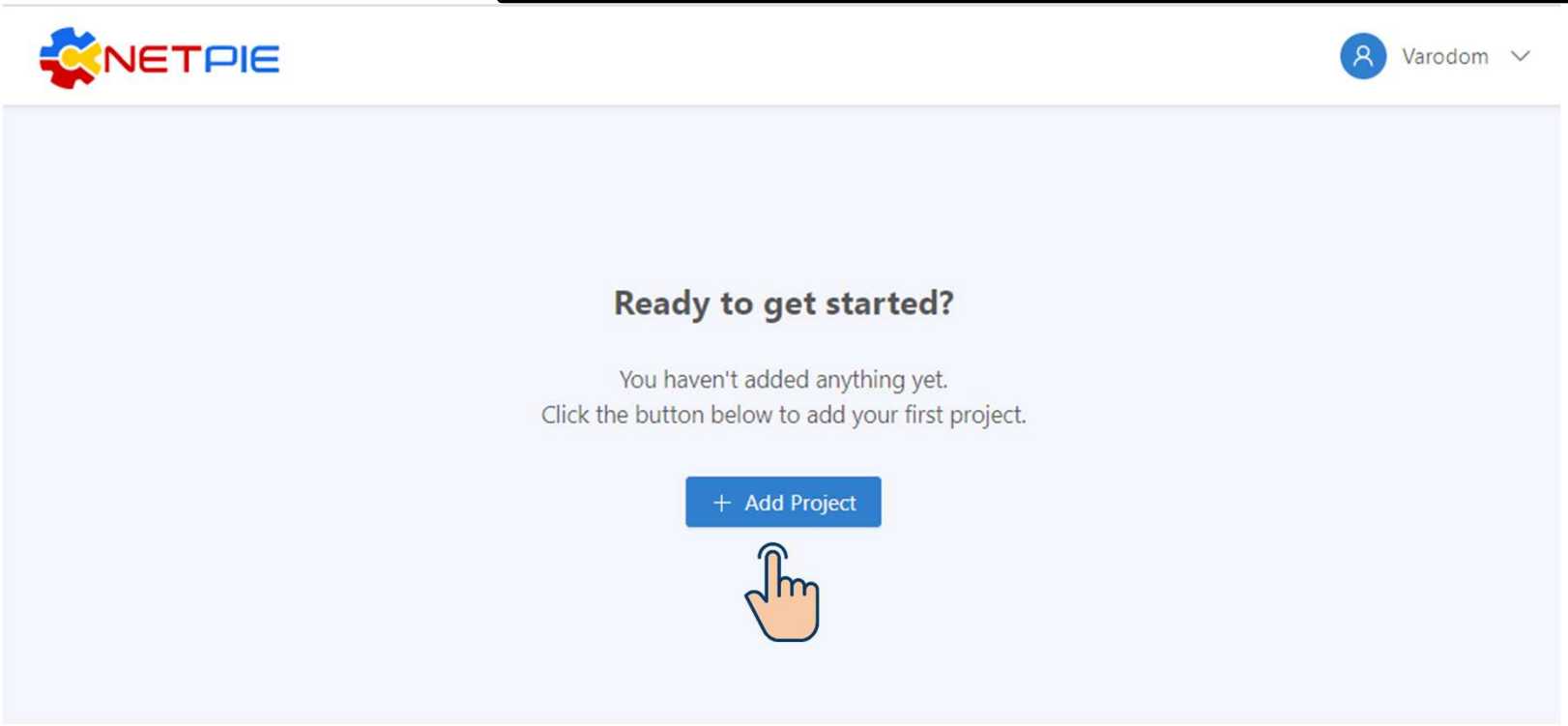


After login, you will see the window as shown

NETPIE 2020 Portal

Create a Project

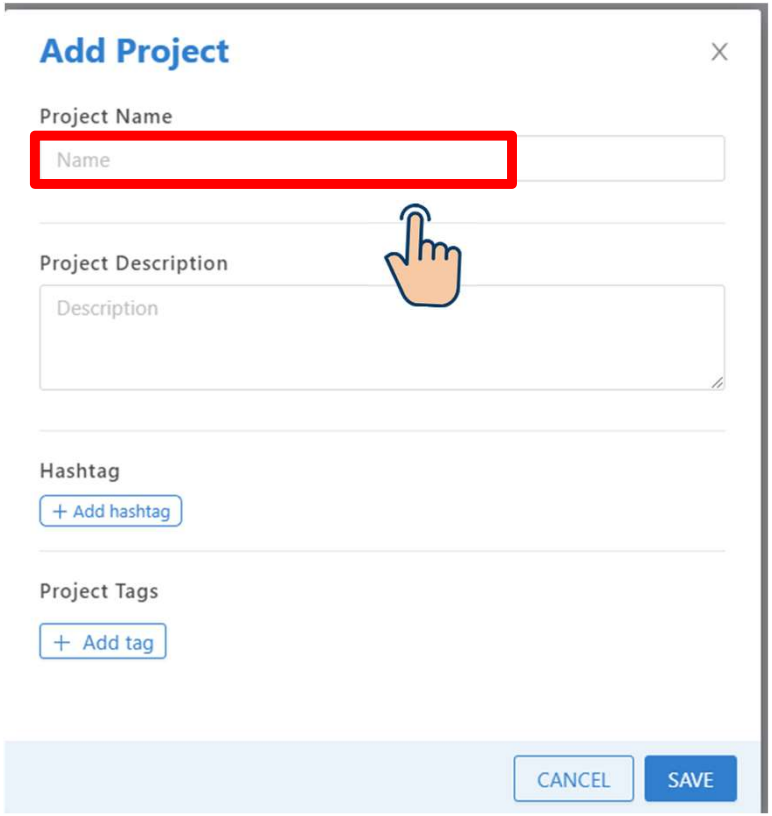
To create a project, click at [+] Add Project as shown in the picture



NETPIE 2020 Portal

Create a Project

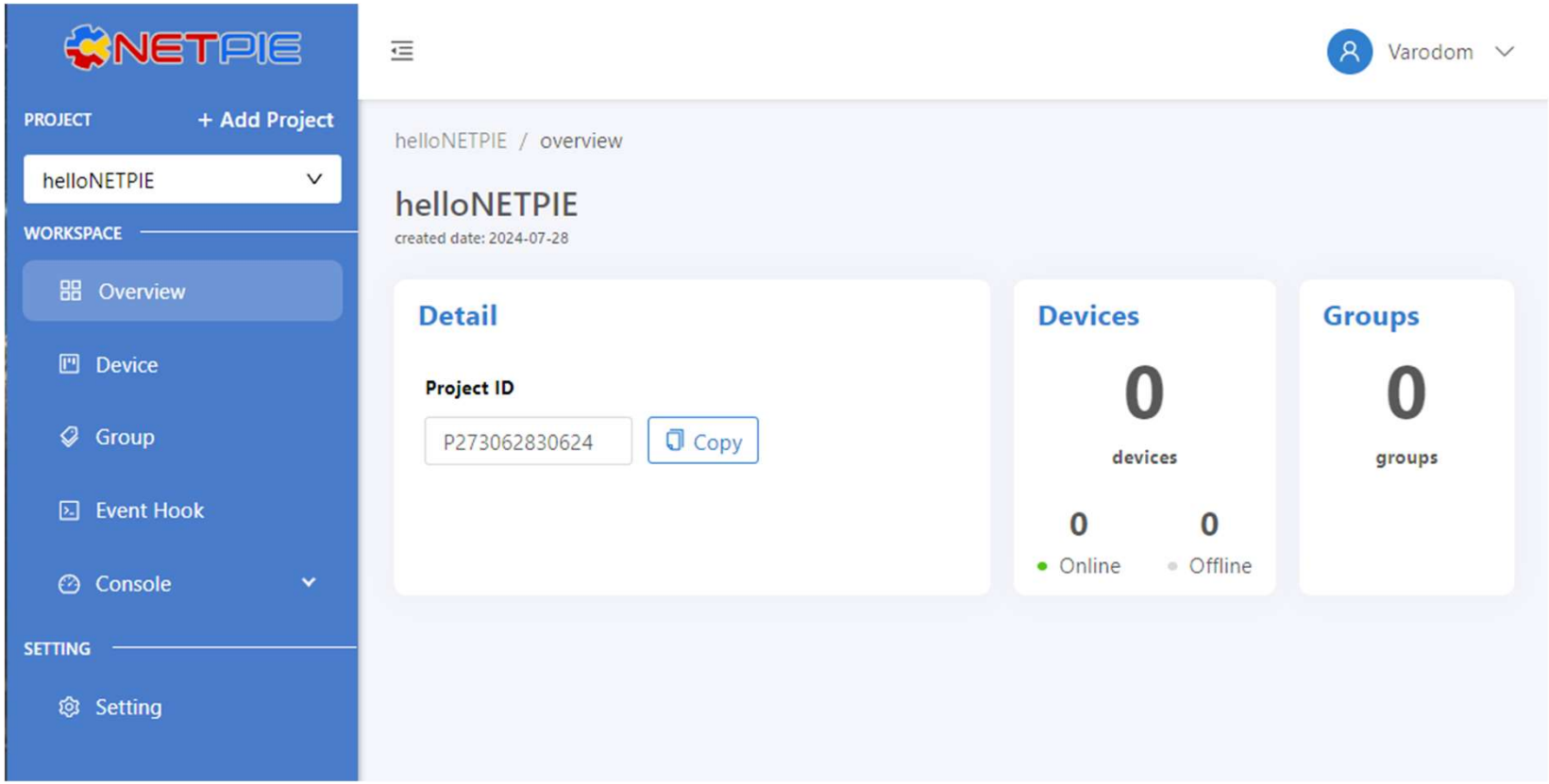
Fill out the information for creating the project.
The Project Name field is required.



NETPIE 2020 Portal

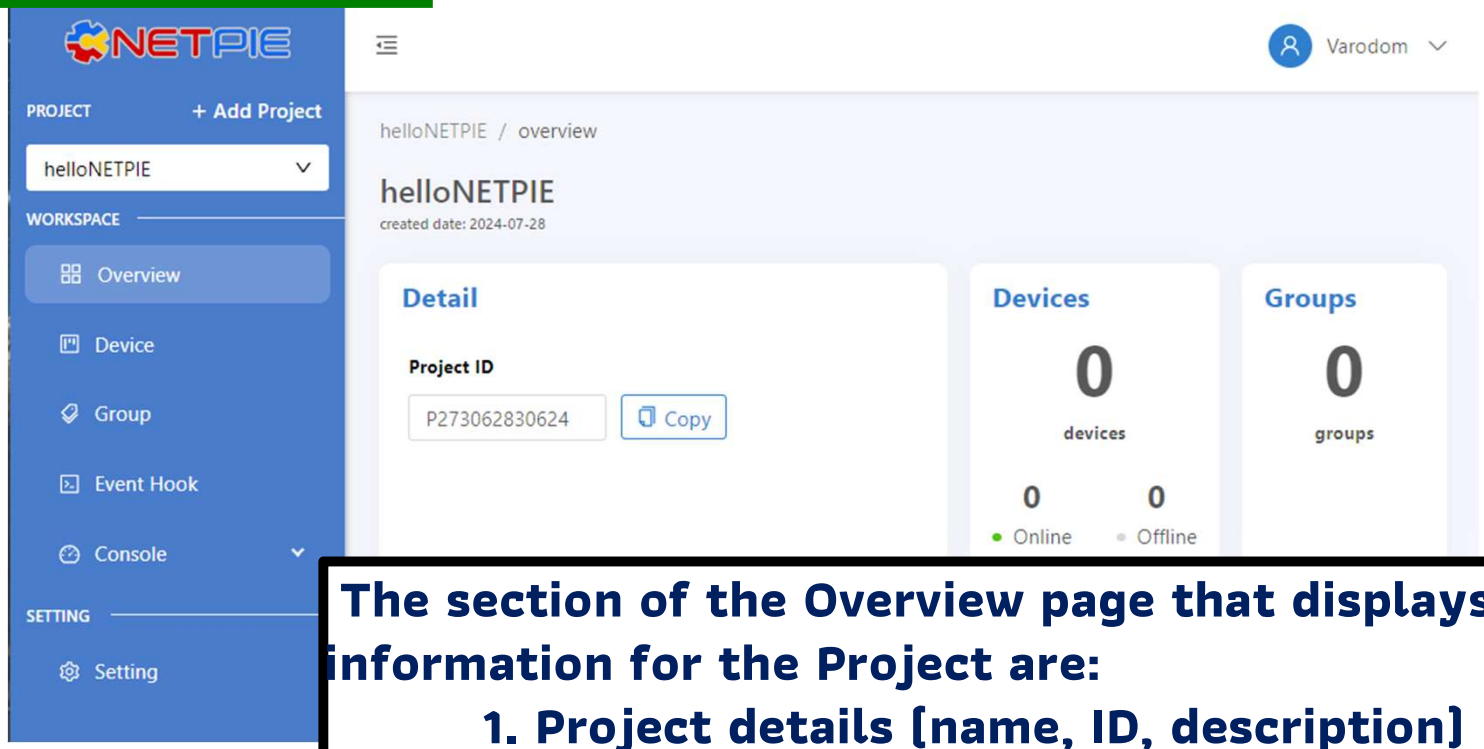
Create a Project

After creating a project, you'll see the first page .



NETPIE 2020 Portal

Create a Project

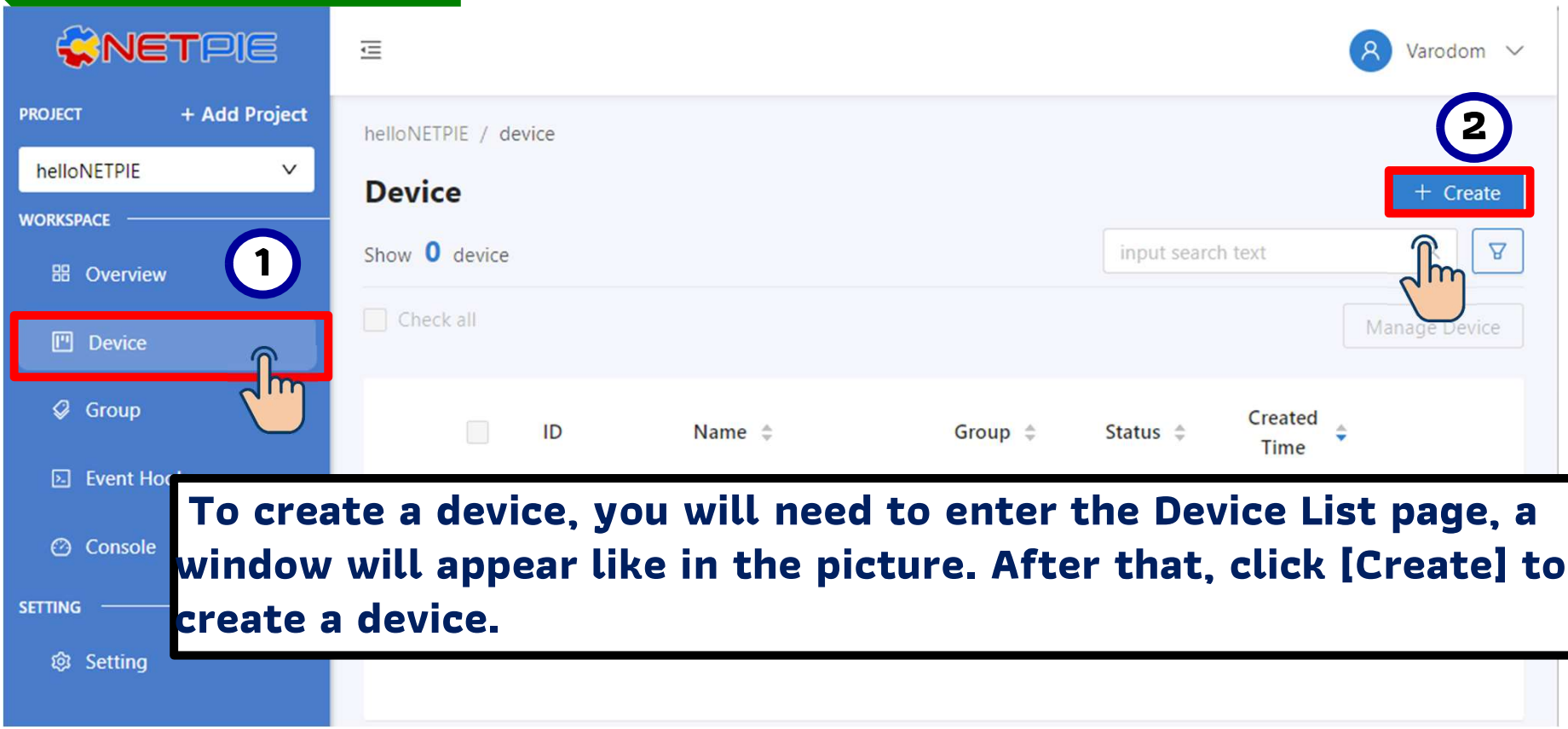


The section of the Overview page that displays all the information for the Project are:

1. **Project details [name, ID, description]**
2. **Devices [Device Online/Offline]**
3. **Groups**

NETPIE 2020 Portal

Device Creation

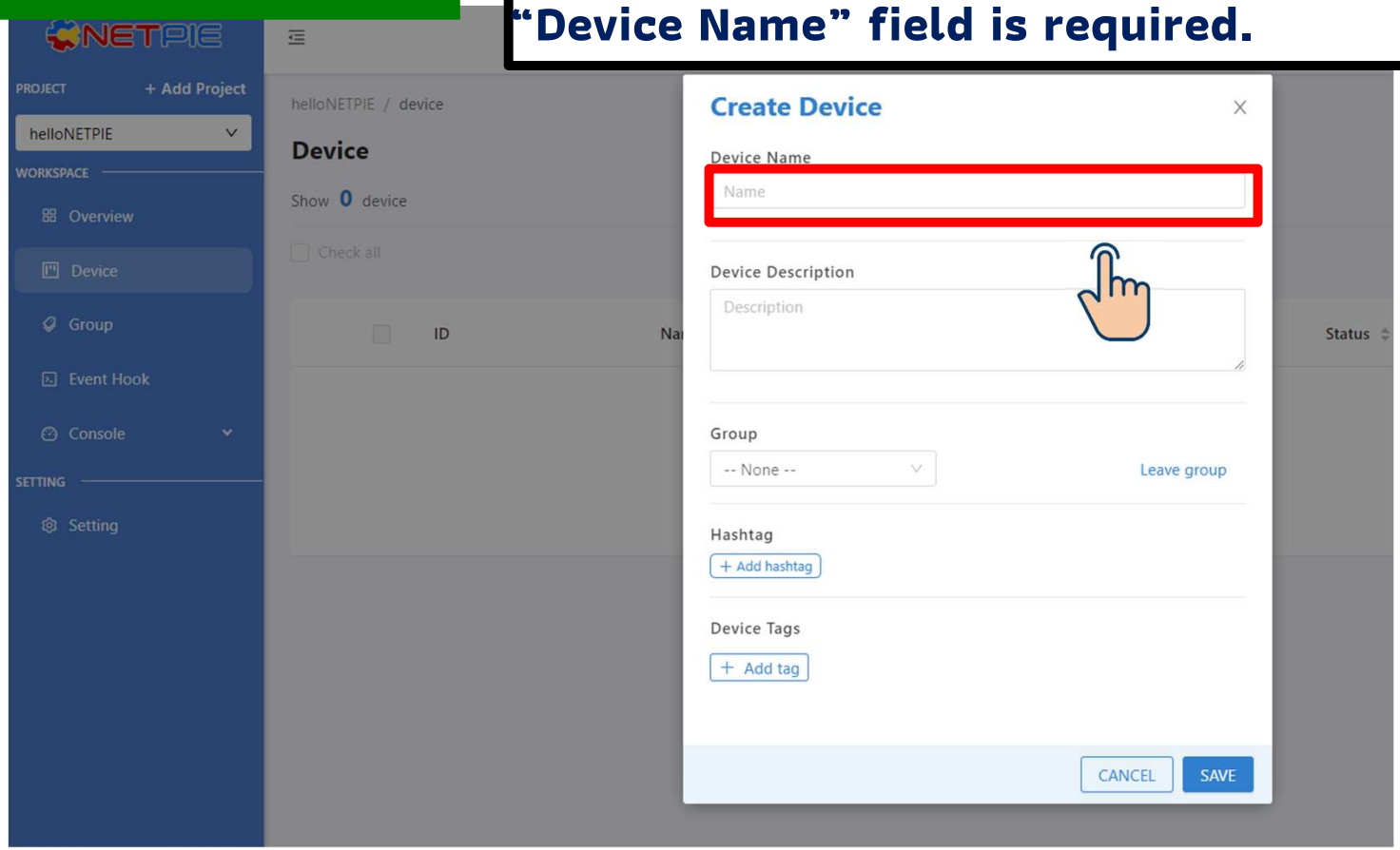


To create a device, you will need to enter the Device List page, a window will appear like in the picture. After that, click [Create] to create a device.

NETPIE 2020 Portal

Device Creation

Fill out the information for creating the device. The "Device Name" field is required.



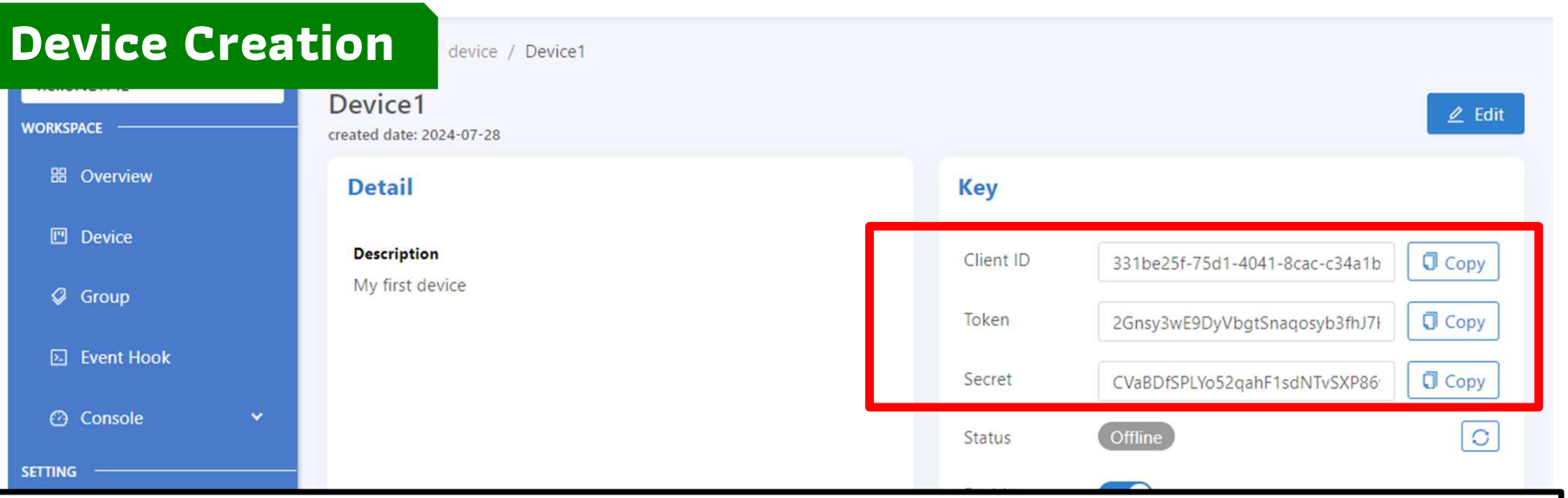
NETPIE 2020 Portal

Device Creation

After creating the device, click the Device tab to view the device information.

NETPIE 2020 Portal

Device Creation



After entering the Device, you will find the details of the Device, that is

1. **Client ID**
2. **Token**
3. **Secret**

These 3 parameters are important to connect the device to NETPIE2020 using various protocols.

Workshop 1 : connecting NETPIE2020 with MQTT clients

MQTTBox

MQTT Protocol

MQTTX

NETPIE2020

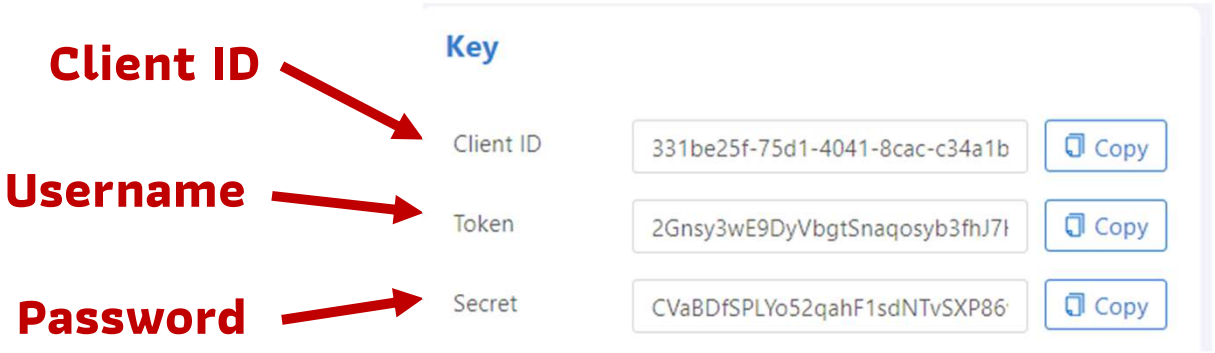
Note : MQTTBox is no longer supported by Chrome. MQTTX, an MQTT client alternative, will be used in class. MQTTBox slides are kept for reference only.

Workshop 1 : connecting NETPIE2020 with MQTTBox

NETPIE2020 connection via MQTT, a small and popular protocol for M2M communication, can use any MQTT Library compatible with the active device.

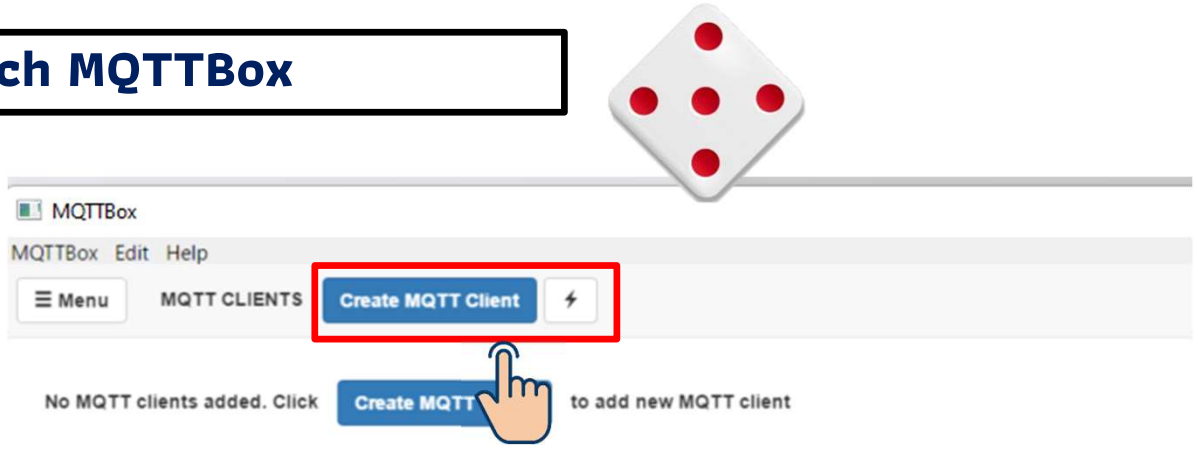
MQTT connection requires 4 Parameters

1. **Host : broker.netpie.io**
2. **Client ID : Client ID of Device created in NETPIE2020**
3. **Username : Token of Device created in NETPIE2020**
4. **Password : Secret of Device created in NETPIE2020 (used for more identity verification)**



Workshop 1 : connecting NETPIE2020 with MQTTBox

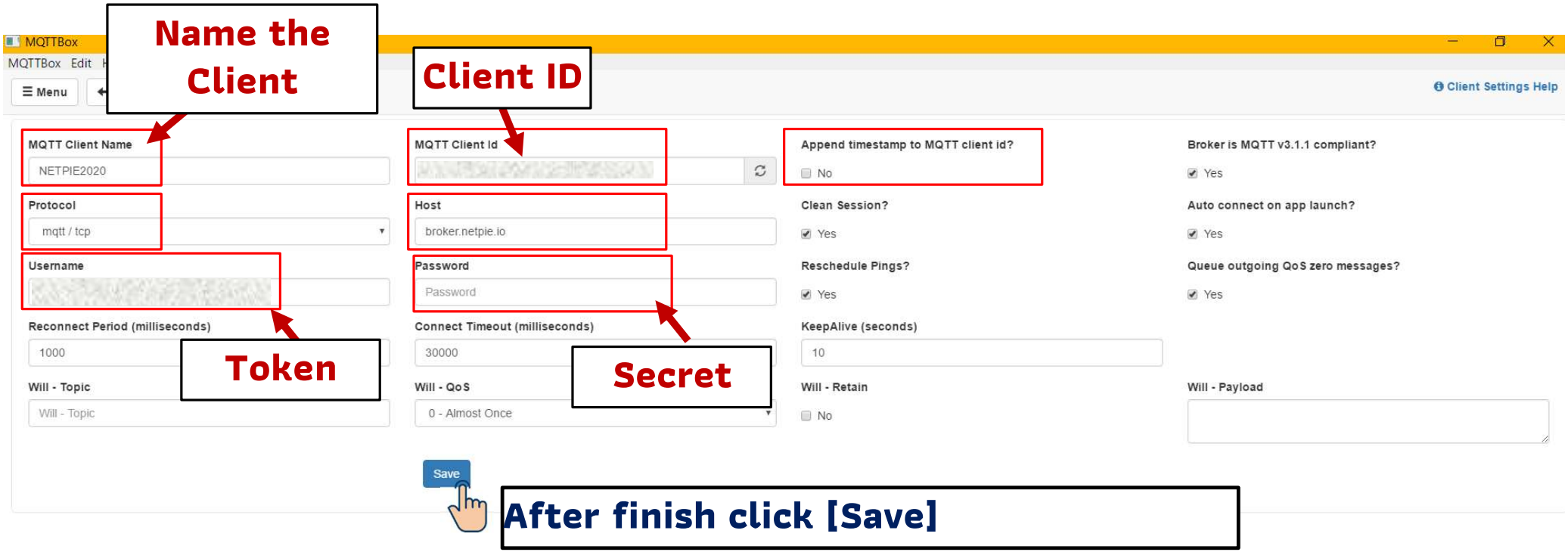
1. Launch MQTTBox



2. Click on [Create MQTT Client]

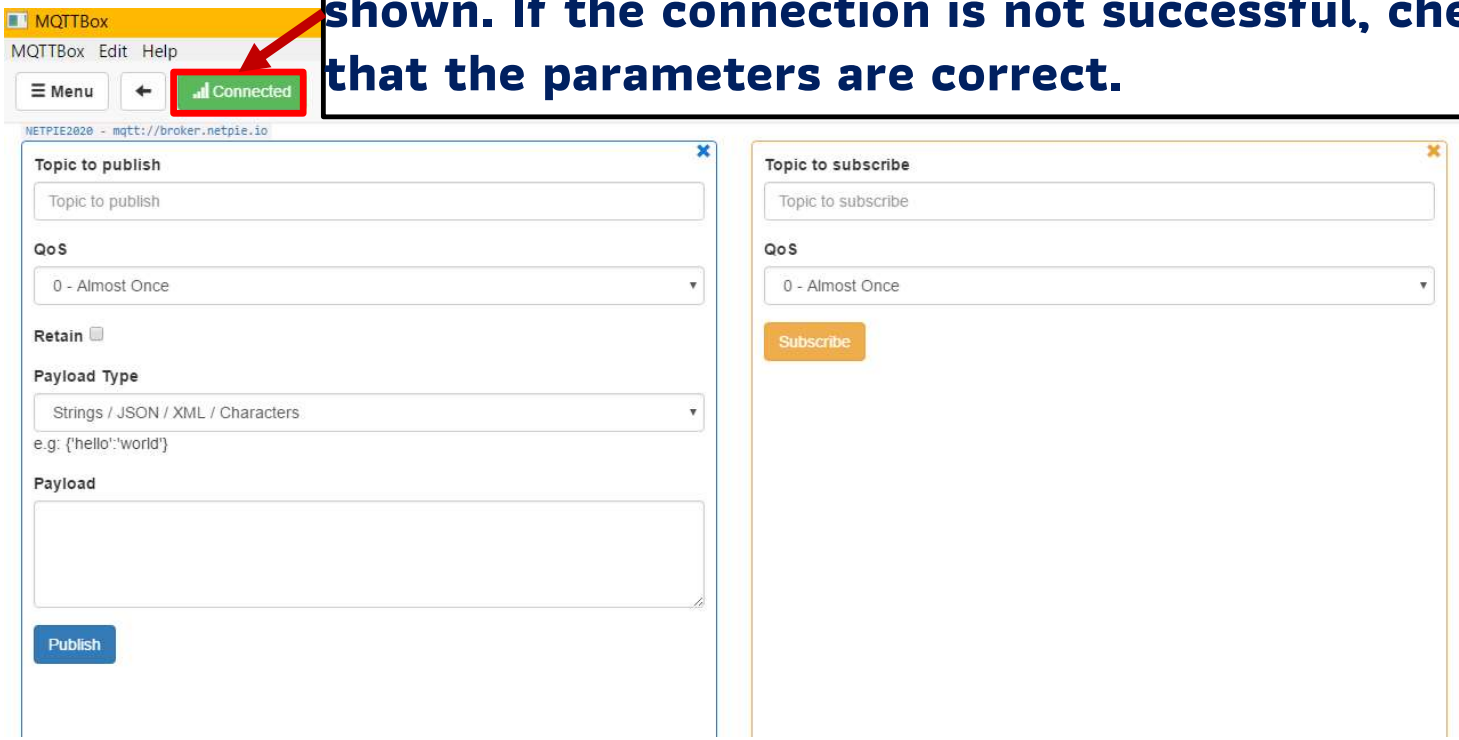
NETPIE Portal

Workshop 1 : connecting NETPIE2020 with MQTTBox



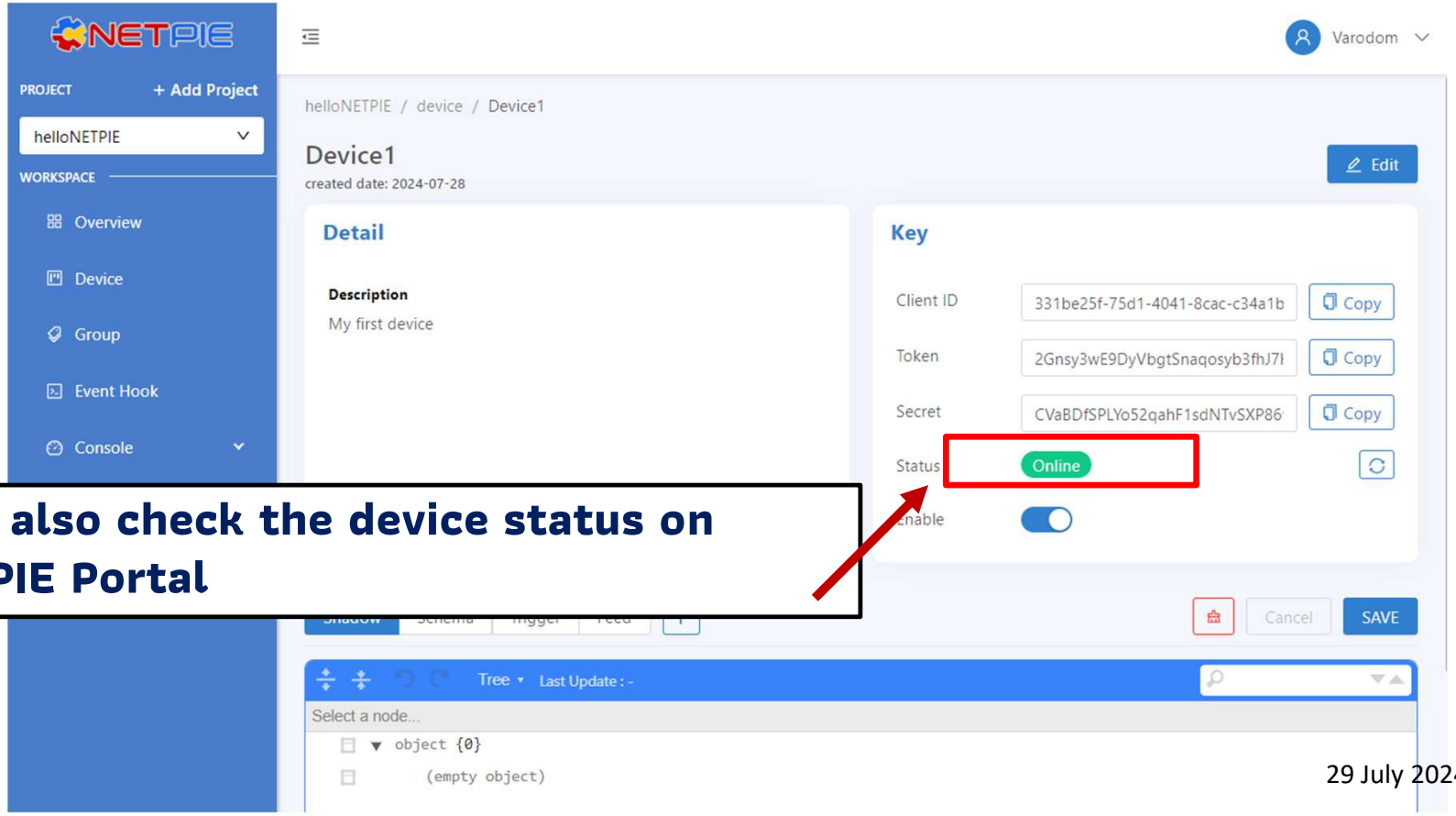
Workshop 1 : connecting NETPIE2020 with MQTTBox

When connected successfully, it will display the results as shown. If the connection is not successful, check to make sure that the parameters are correct.



NETPIE Portal

Workshop 1 : connecting NETPIE2020 with MQTTBox

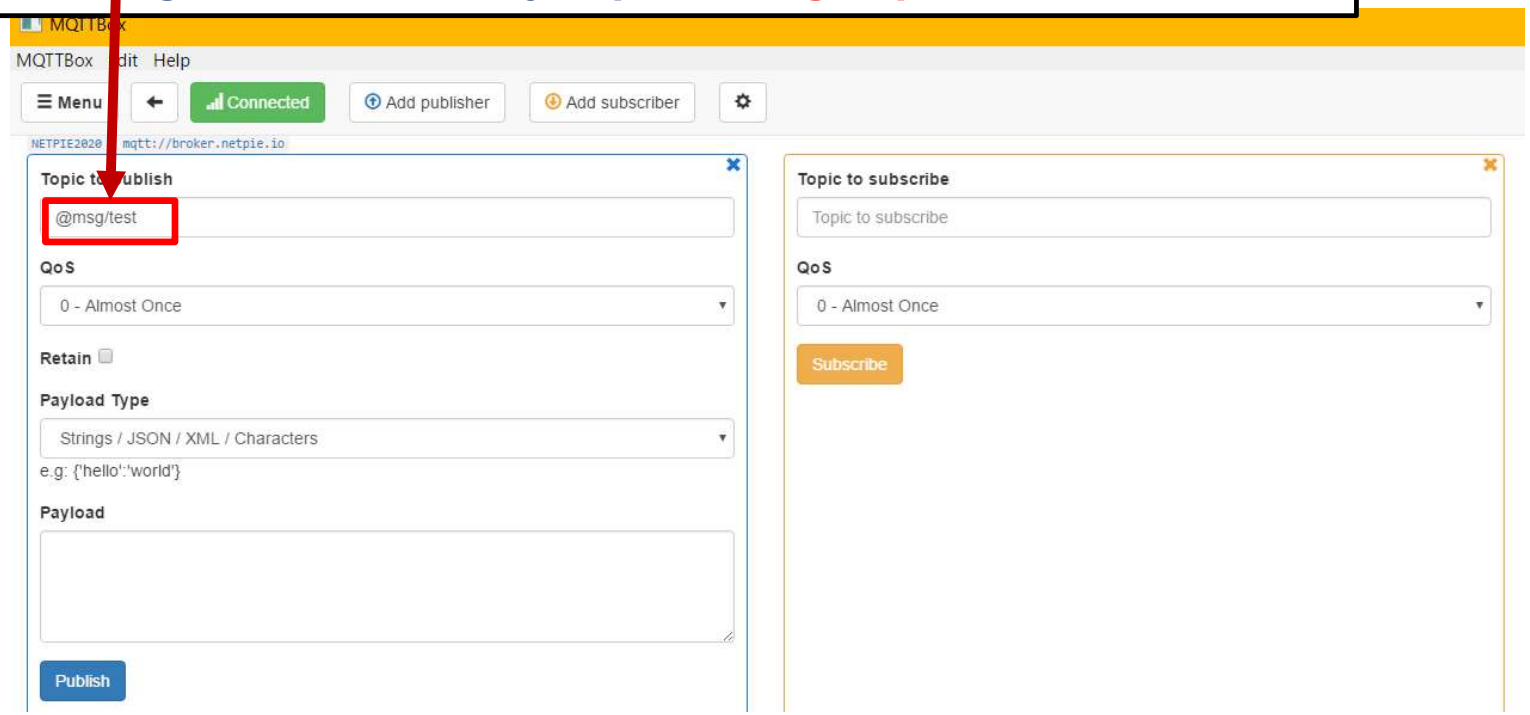


Can also check the device status on NETPIE Portal

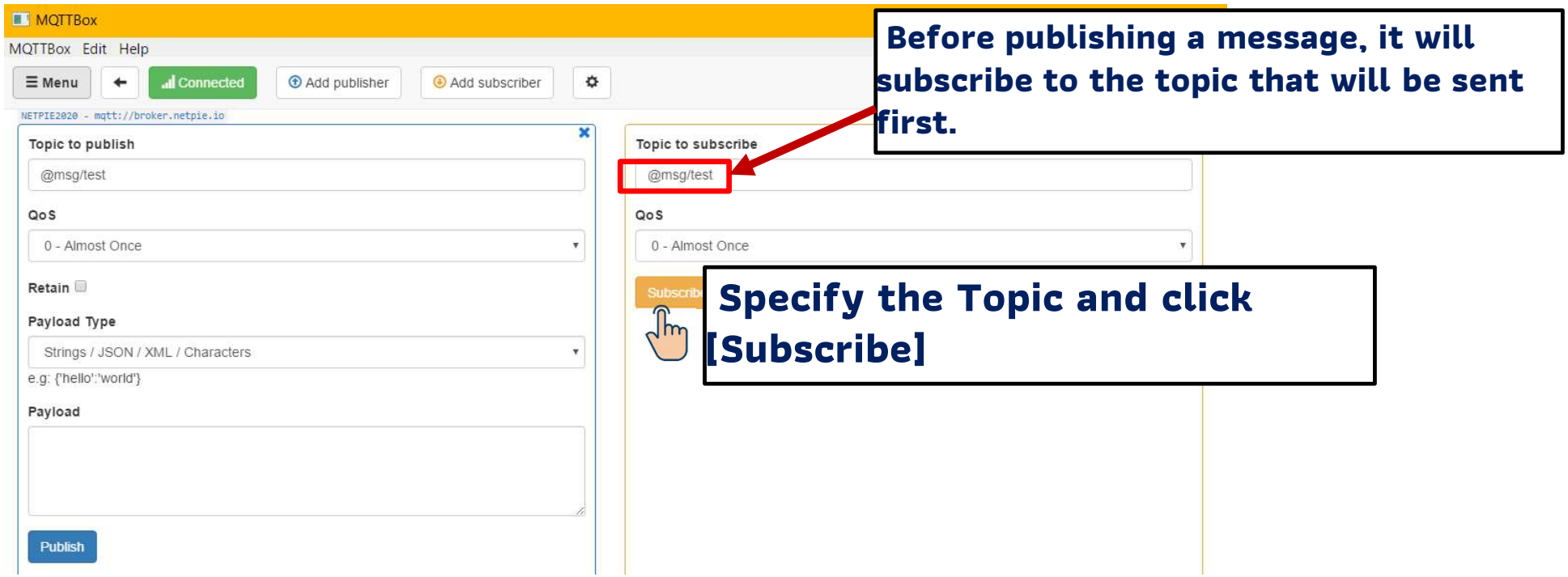
Workshop 1 : connecting NETPIE2020 with MQTTBox

After connection is established, test by sending message to NETPIE2020

MQTT message is referenced by Topic : @msg/topic

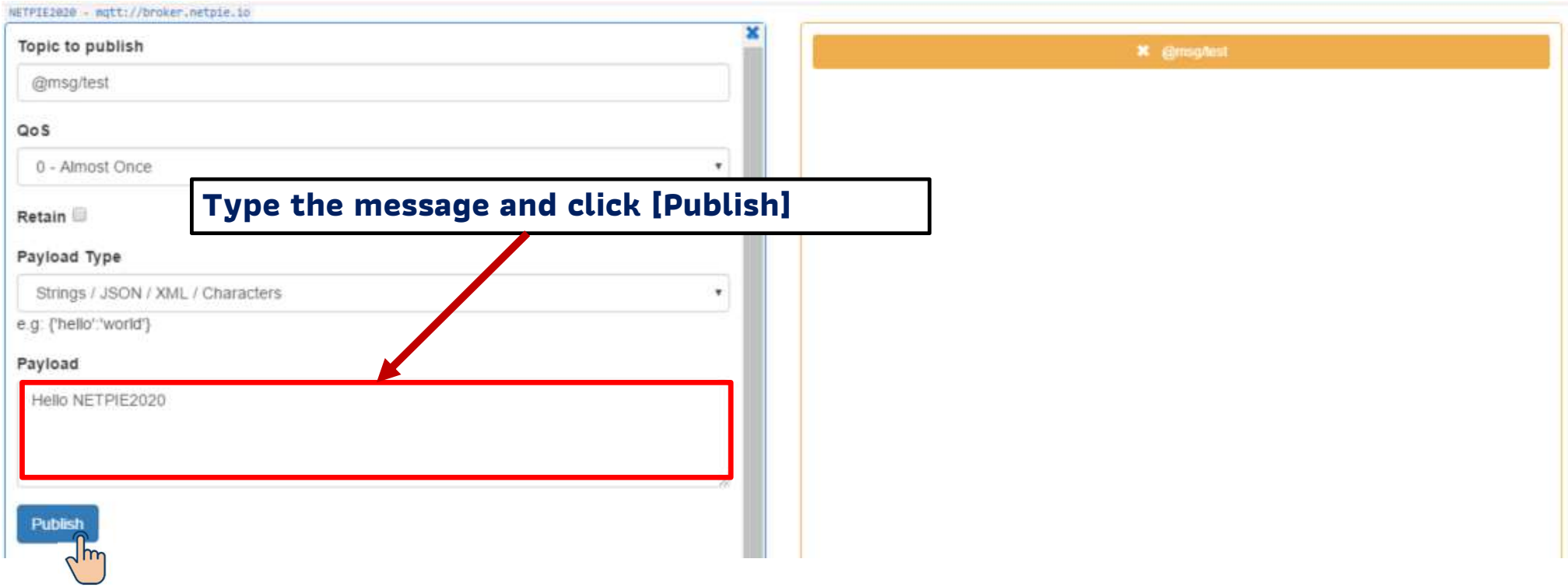


Workshop 1 : connecting NETPIE2020 with MQTTBox



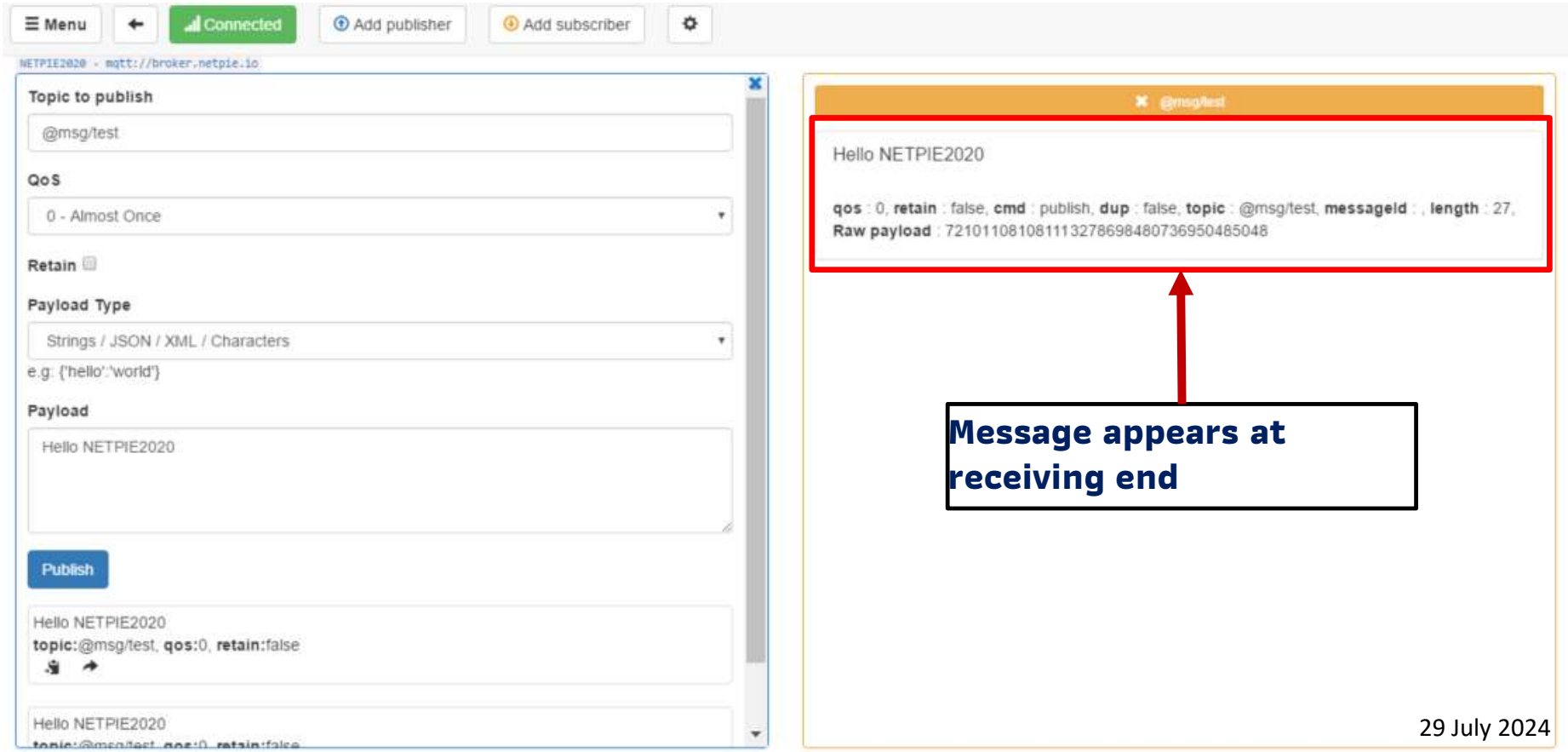
NETPIE Portal

Workshop 1 : connecting NETPIE2020 with MQTTBox



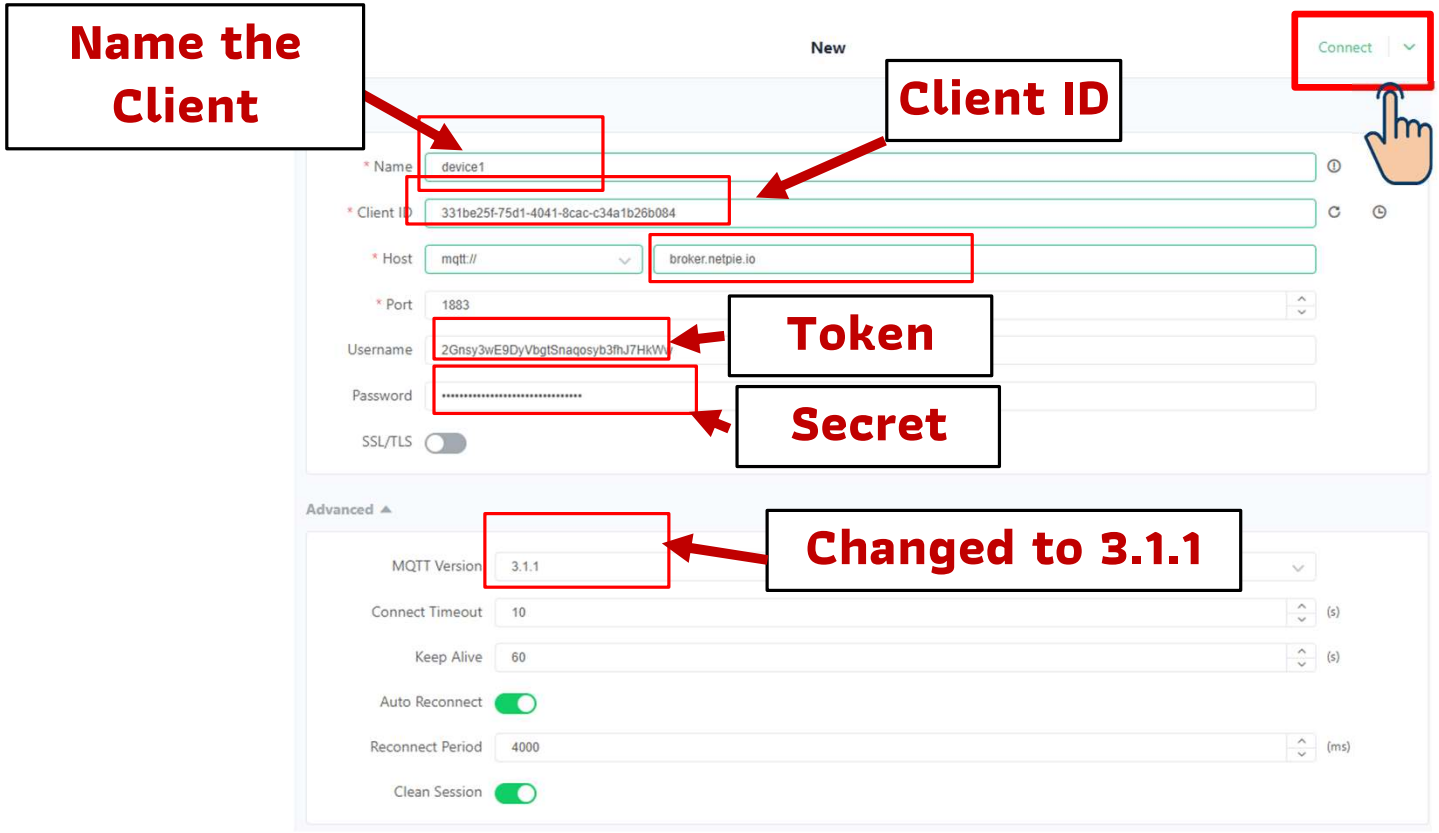
NETPIE Portal

Workshop 1 : connecting NETPIE2020 with MQTTBox

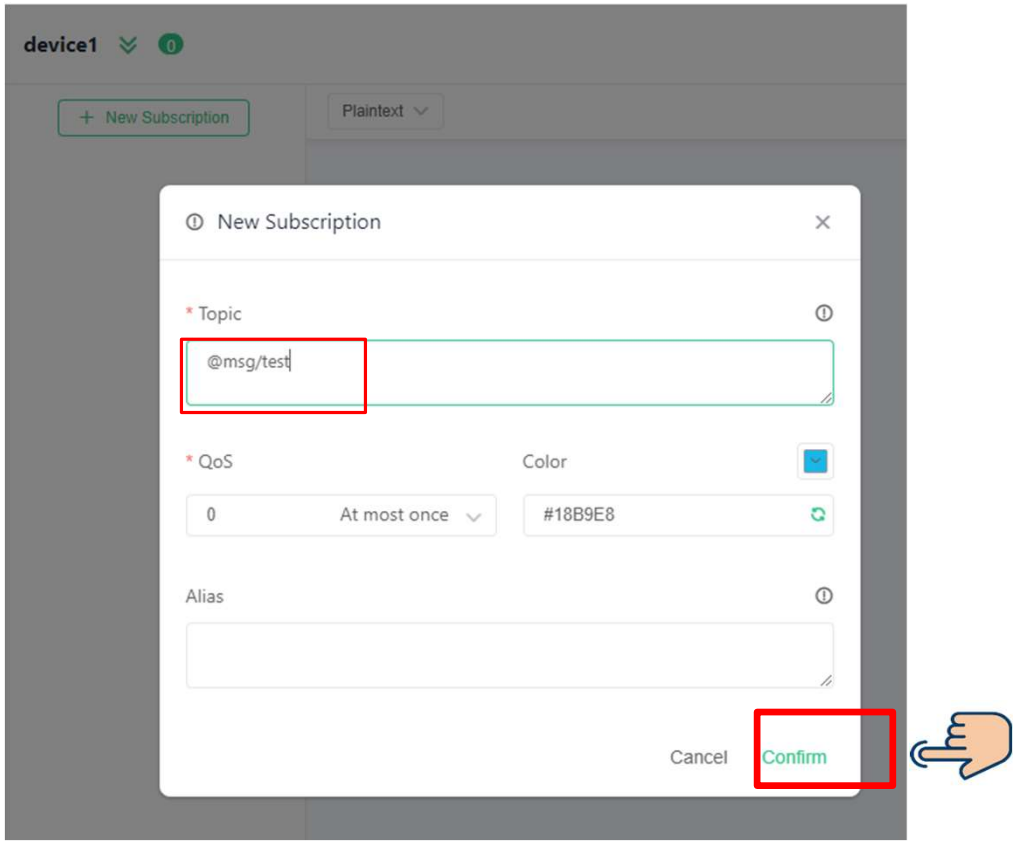


NETPIE Portal

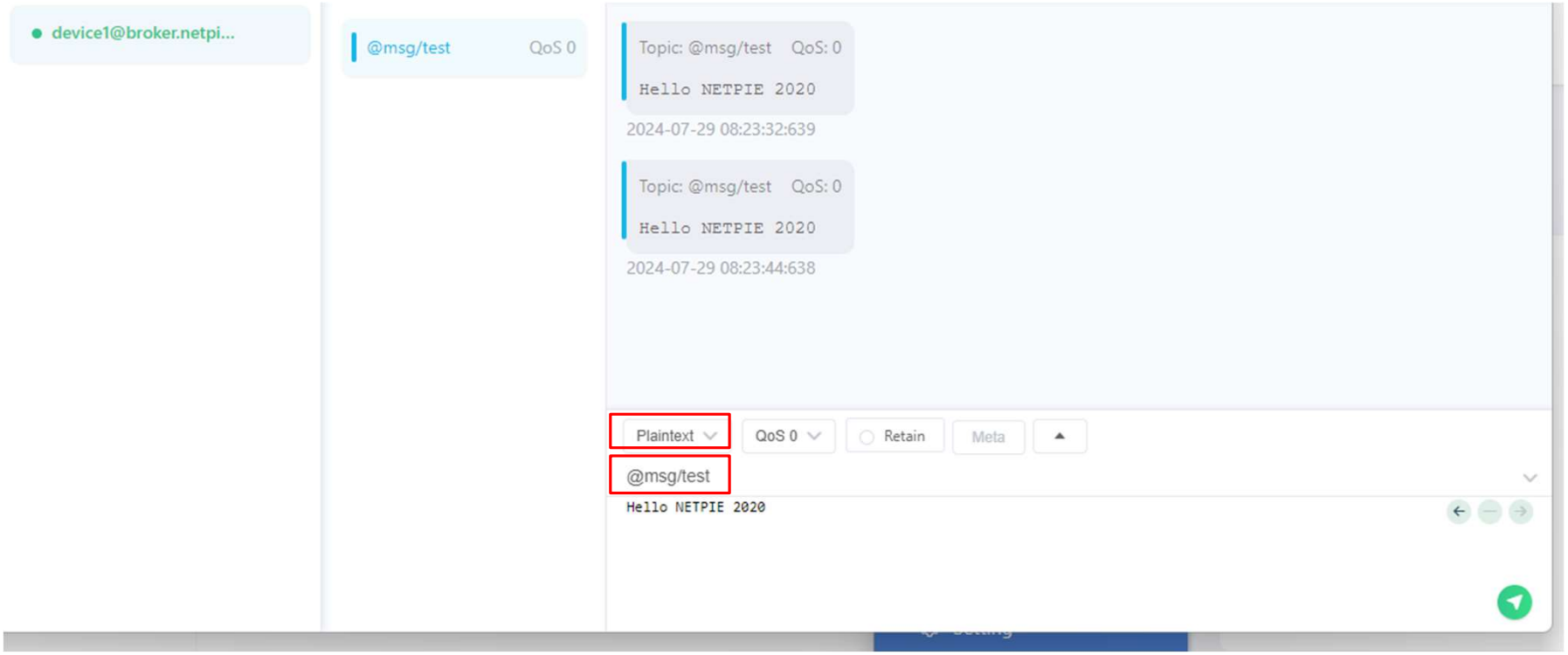
Workshop 1 : connecting NETPIE2020 with MQTTX



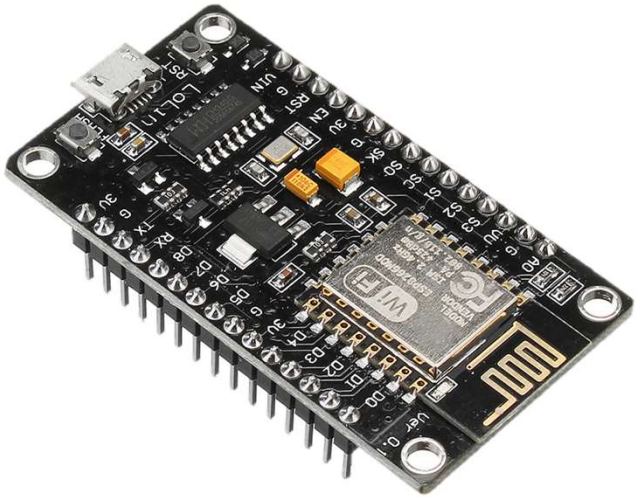
Workshop 1 : connecting NETPIE2020 with MQTTX



Workshop 1 : connecting NETPIE2020 with MQTTX



Workshop 2 : Connect NETPIE2020 with NodeMCU



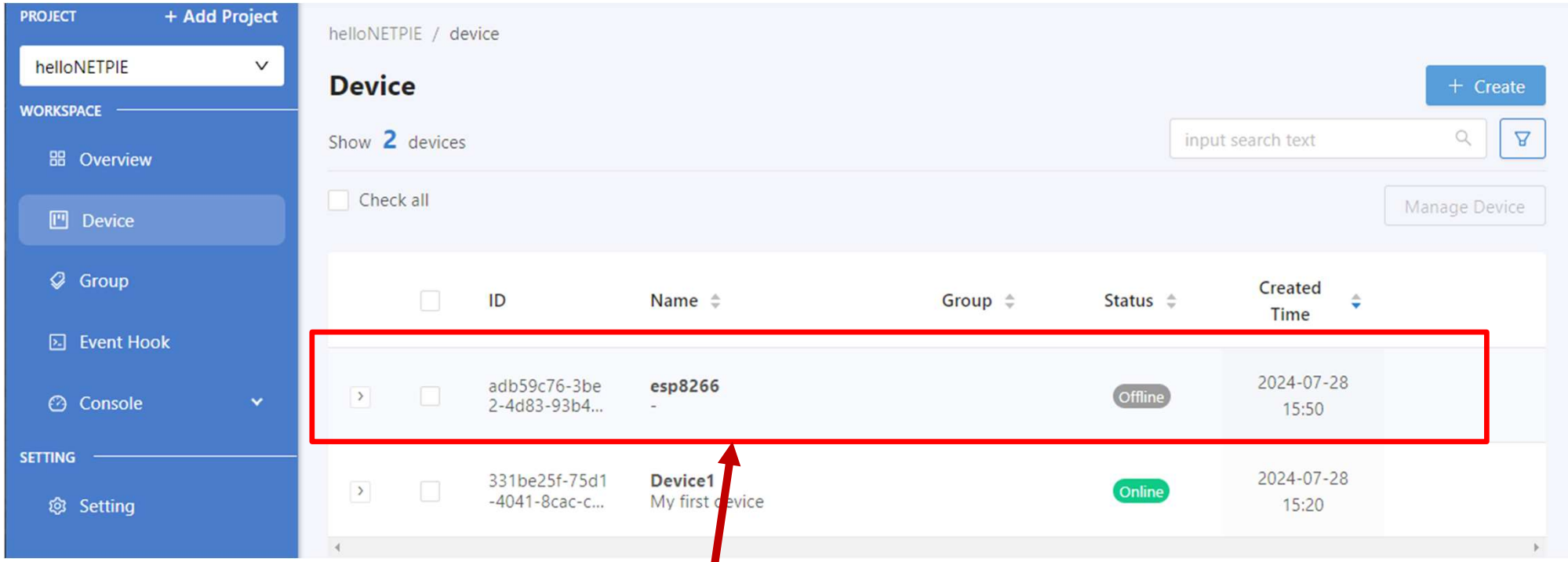
NodeMCU

MQTT Protocol

NETPIE2020

NETPIE Portal

Workshop 2 : Connect NETPIE2020 with NodeMCU



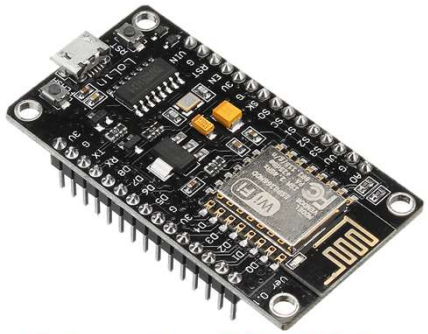
Create new device for NodeMCU

Module 2 : NETPIE 2020 Portal

Workshop 2 : Connect NETPIE2020 with NodeMCU

In order for NodeMCU to connect to NETPIE2020, it needs to be programmed on the Arduino IDE to connect NodeMCU to internet and then to NETPIE2020.

Arduino IDE

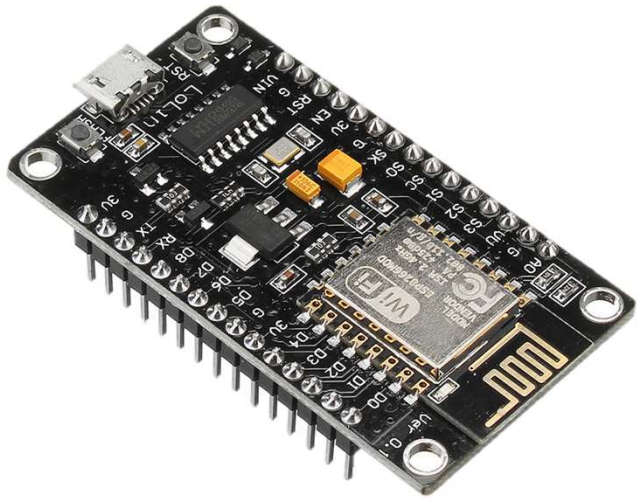


NodeMCU

Internet connection through WiFi

NETPIE2020

Workshop 2 : Connect NETPIE2020 with NodeMCU



NodeMCU

In order for NodeMCU to connect to NETPIE2020, the following libraries are needed

1. ESP8266WiFi

Used to allow NodeMCU to connect to the Internet via WiFi network.

2. PubSubClient

Used for NodeMCU to connect and communicate on NETPIE 2020 platform. (MQTT Protocol)

Workshop 2 : Connect NETPIE2020 with NodeMCU

Key command and functions in ESP8266WiFi

WiFi.begin()

WiFi.begin : function for setup Wi-Fi Library and network, with status as results Usage example:

WiFi.begin();

WiFi.begin(ssid); ssid is the Wi-Fi network to connected to

WiFi.begin(ssid ,pass); pass is the wifi password WPA encrypted

WiFi.status

WiFi.status is a function to check connection with the wifi network. Usage example :

WiFi.status() != WL_CONNECTED

Results from WiFi.begin()

WL_CONNECTED : successful connection
WL_IDLE_STATUS : ready but not connected

Results from WiFi.status()

WL_CONNECTED : successful connection
WL_NO_SHIELD : Wi-Fi Shield not found
WL_DILE_STATUS : ready but not connected. This is a temporary status while connection is attempted.
WL_NO_SSID_AVAIL : no SSID to connect
WL_SCAN_COMPLETED : network scan finished
WL_CONNECT_FAILED : connection not successful
WL_CONNECTION_LOST : lost the connection
WL_DISCONNECTED : disconnected

Workshop 2 : Connect NETPIE2020 with NodeMCU

Important commands & functions in PubSubClient

void callback

callback is a function for receiving payloads or messages sent by the subscribed Topic.

void reconnect

reconnect is a function to connect to the MQTT Server that has been set up. Or when there is a disconnection event with the MQTT Server, the reconnect function will be invoked to reconnect.

client.connect()

client.connect() : command to connect to MQTT Broker. Format is

client.connect(client, username, password)

client.setServer()

client.setServer() : command to setup MQTT Server. Format is

client.setServer(mqtt_server, mqtt_port)

client.publish()

client.publish() : to publish with specified topic

client.publish("topic", "Message")

client.subscribe()

client.subscribe to subscribe specified Topic. Often placed in reconnect function. Format is

client.subscribe("topic")

Workshop 2 : Connect NETPIE2020 with NodeMCU

Open Workshop2.ino

Coding in Workshop2.ino consists of 3 sections

1. **Section 1 : Library and variable declaration**
   - **Call to various libraries.**
   - **Variable declaration for connecting to WiFi and for NETPIE2020 connection**
   - **Create instances for NETPIE2020 connection**
   - **Global variables**

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

const char* ssid = "Your_SSID";
const char* password = "Your_Password";
const char* mqtt_server = "broker.netpie.io";
const int mqtt_port = 1883;
const char* mqtt_Client = "Client_ID";
const char* mqtt_username = "Token";
const char* mqtt_password = "Secret";

WiFiClient espClient;
PubSubClient client(espClient);

long lastMsg = 0;
int value = 0;
```

Workshop 2 : Connect NETPIE2020 with NodeMCU

2 **Section 2 : functions**

Functions to connect to MQTT

```
void reconnect() {
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    if (client.connect(mqtt_Client, mqtt_username, mqtt_password)) {
      Serial.println("connected");
    }
    else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println("try again in 5 seconds");
      delay(5000);
    }
  }
}
```

Enter NETPIE2020 MQTT connection.

- **If connected successfully, it will display "connected".**
- **If the connection is not successful, it will display “failed…” and automatically reconnect**

Workshop 2 : Connect NETPIE2020 with NodeMCU

2 **Section 2 : functions**

Performing initialization

```
void setup() {
  Serial.begin(115200);
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
  client.setServer(mqtt_server, mqtt_port);
}
```

In this function, it will connect to WiFi and NETPIE2020 according to the settings

Workshop 2 : Connect NETPIE2020 with NodeMCU

3 Section 3 : loop[] function

Main function that runs iteratively

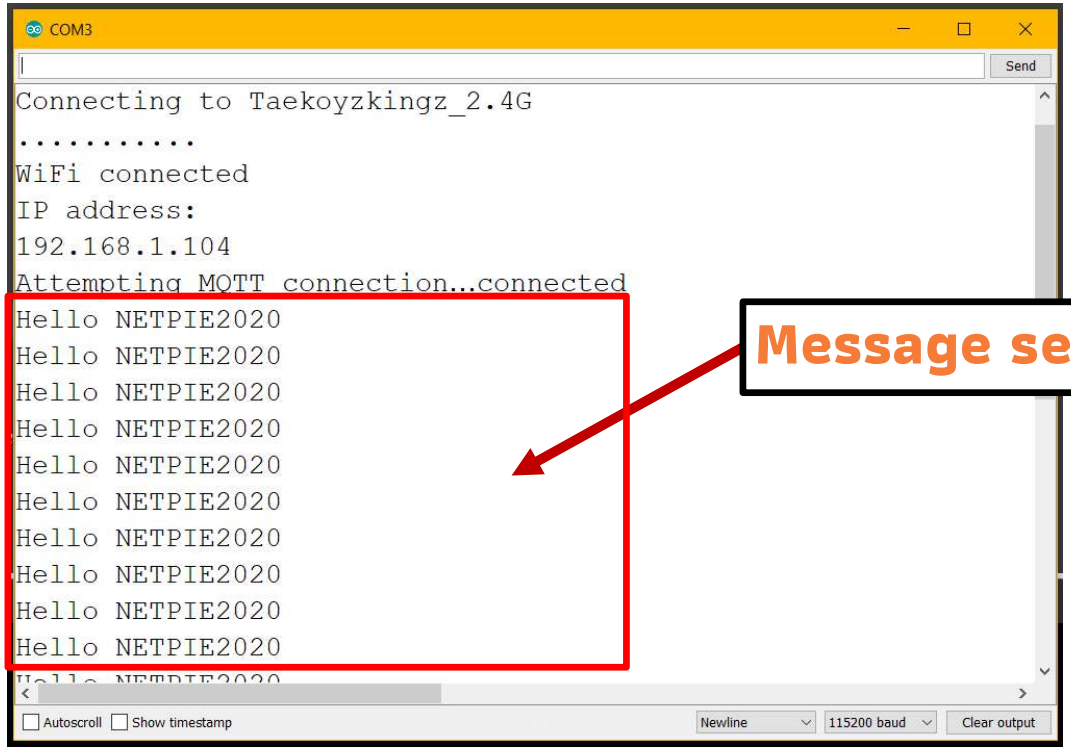
Command set that maintains the connection status and functions of NETPIE2020.

Commands to send message “Hello NETPIE2020” to Topic : @msg/test every 2 seconds

```
void loop() {
  if (!client.connected()) {
    reconnect();
  }
  client.loop();

  long now = millis();
  if (now - lastMsg > 2000) {
    lastMsg = now;
    ++value;
    client.publish("@msg/test", "Hello NETPIE2020");
    Serial.println("Hello NETPIE2020");
  }
  delay(1);
}
```

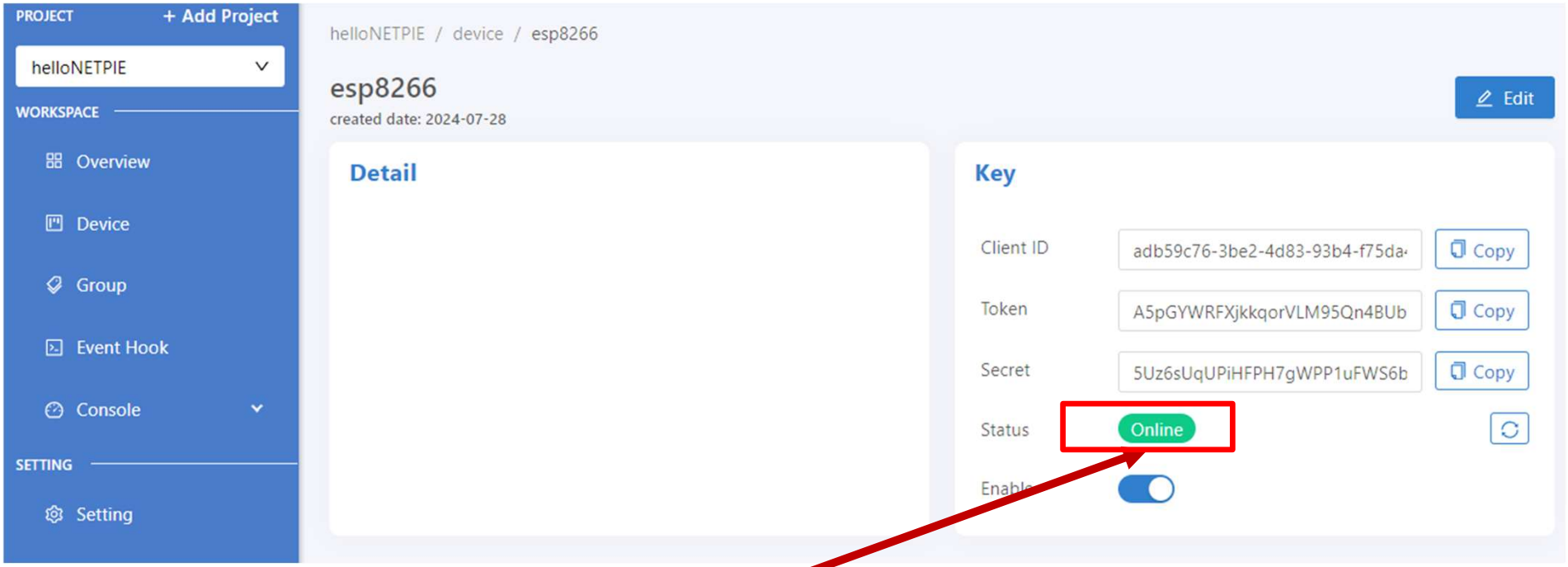
Workshop 2 : Connect NETPIE2020 with NodeMCU



Message sent to NETPIE2020

NETPIE Portal

Workshop 2 : Connect NETPIE2020 with NodeMCU



Check NodeMCU status on NETPIE Portal

Displaying message NodeMCU sent to NETPIE2020

To display message from NodeMCU can be done by bringing another Device/App to connect vis MQTT Protocol And subscribing topic published by NodeMCU Using same Client-ID, Token and Secret

NodeMCU

Authentication
Client-ID : 13456789
Token : abcdefghijk
Secret : lmnopqrsty

Publish "Hello NETPIE2020"

NETPIE2020

Displaying message NodeMCU sent to NETPIE2020

Example

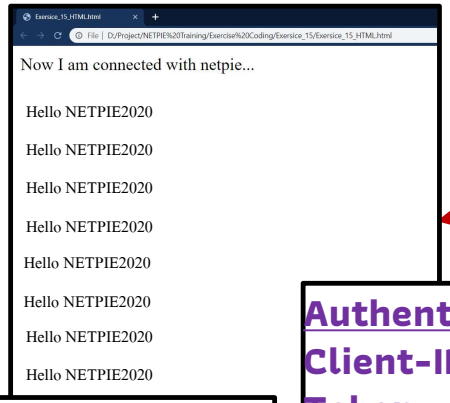
NodeMCU

Authentication
Client-ID : 13456789
Token : abcdefghijk
Secret : lmnopqrsty

Publish
"Hello NETPIE2020"

NETPIE2020

Subscribe
"Hello NETPIE2020"



Web Browser

Authentication
Client-ID : 13456789
Token : abcdefghijk
Secret : lmnopqrsty

Displaying message NodeMCU sent to NETPIE2020

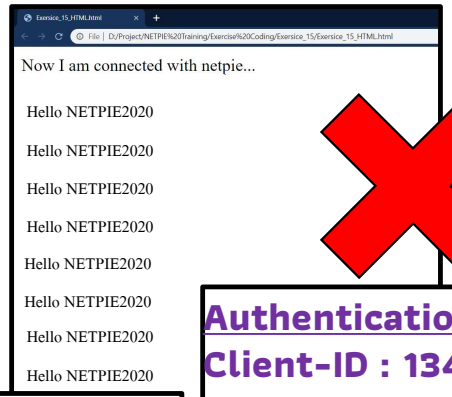
NodeMCU

Authentication
Client-ID : 13456789
Token : abcdefghijk
Secret : lmnopqrsty

Publish "Hello NETPIE2020"

Subscribe "Hello NETPIE2020"

NETPIE2020



Web Browser

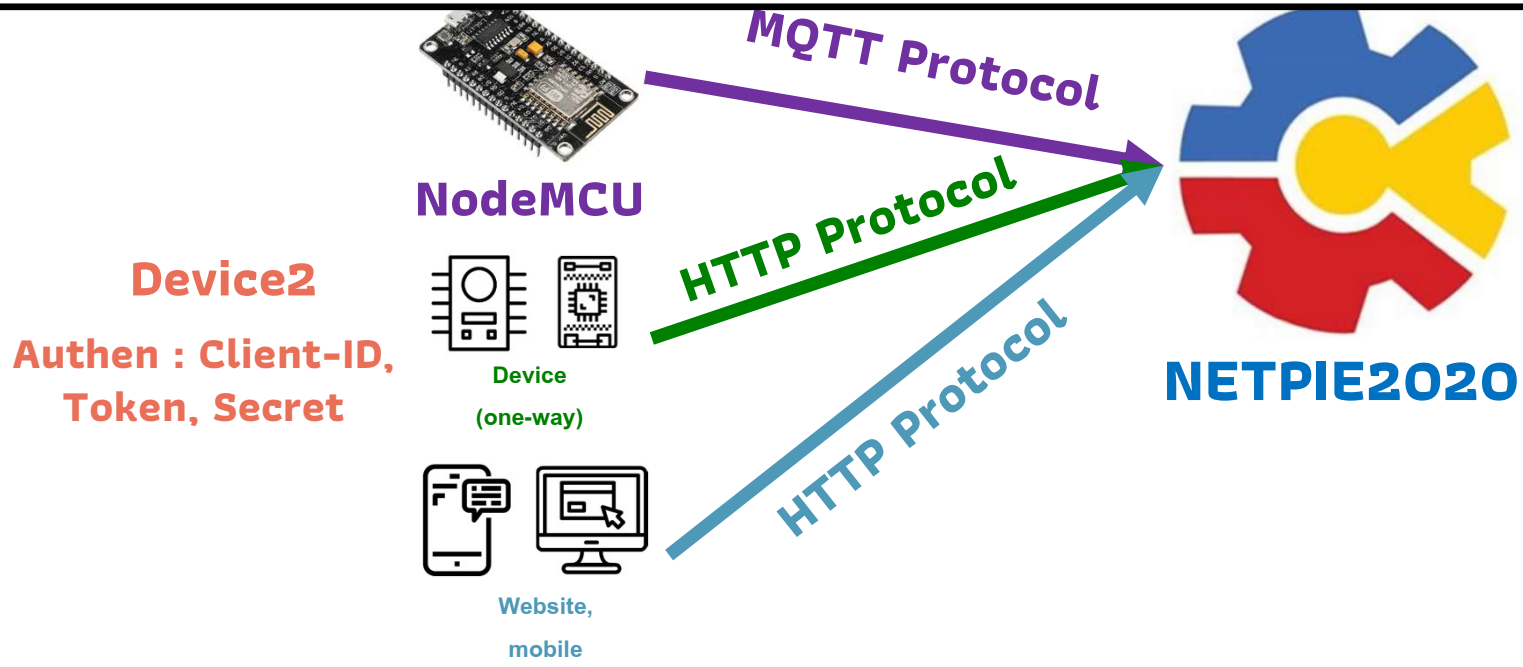
Authentication
Client-ID : 13456789
Token : abcdefghijk
Secret : lmnopqrsty

An example to connect Web Browser with NETPIE2020 via MQTT Protocol using same Client-ID, Token, Secret as NodeMCU

This in effect prevents the Web Browser from connecting with NETPIE2020 since the Client-ID, Token, Secret are already used

Device connection with MQTT Protocol

If you want to use the same device to connect, you must select another protocol.



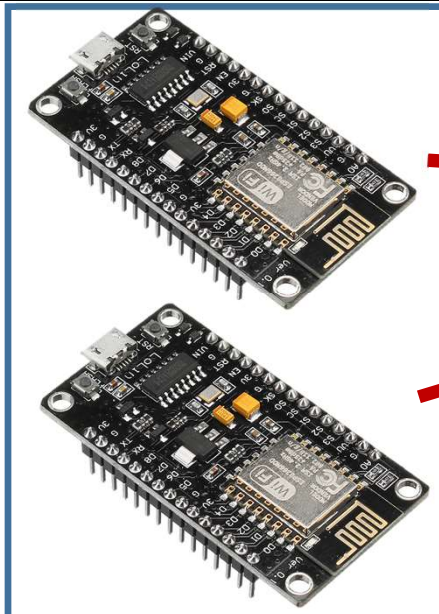
NETPIE Portal

Device connection with MQTT Protocol

If you want to see messages that are sent using the MQTT Protocol, you must create additional devices and place them in the same group.

NodeMCU
Device2
Authen : Client-ID, Token, Secret

NodeMCU
Device1
Authen : Client-ID, Token, Secret



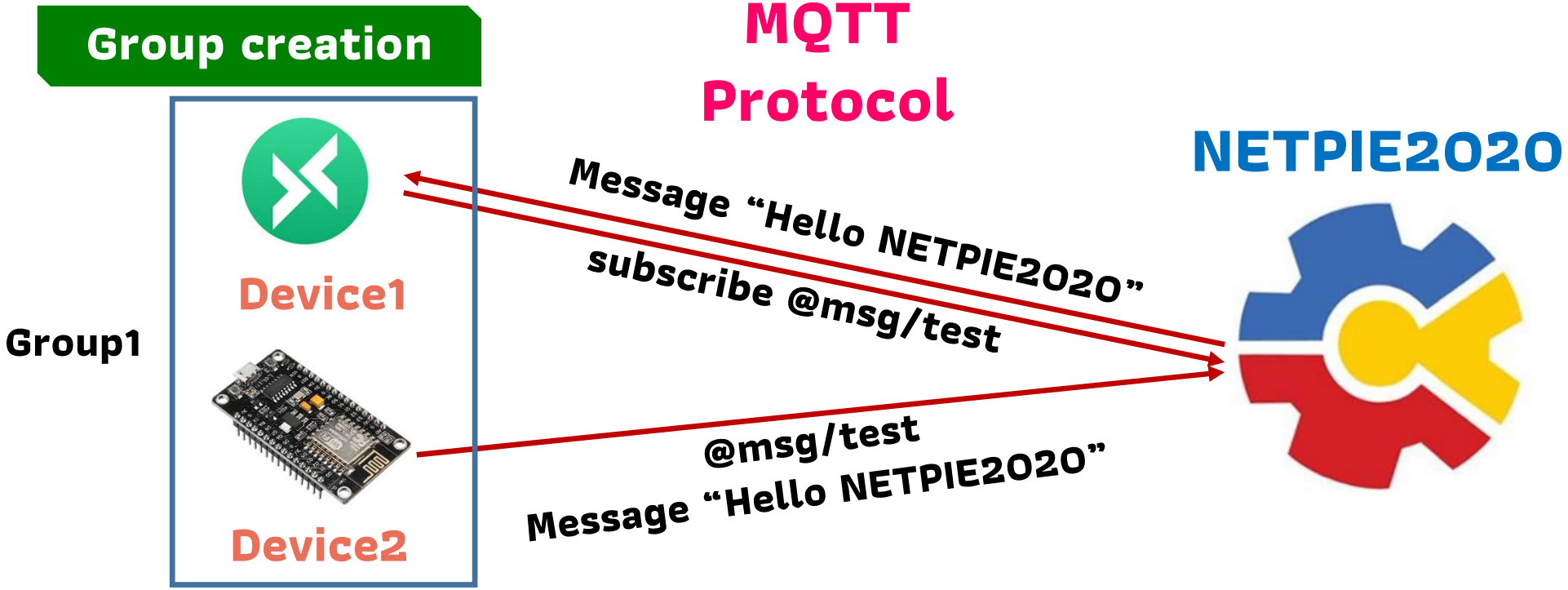
Group1

MQTT Protocol

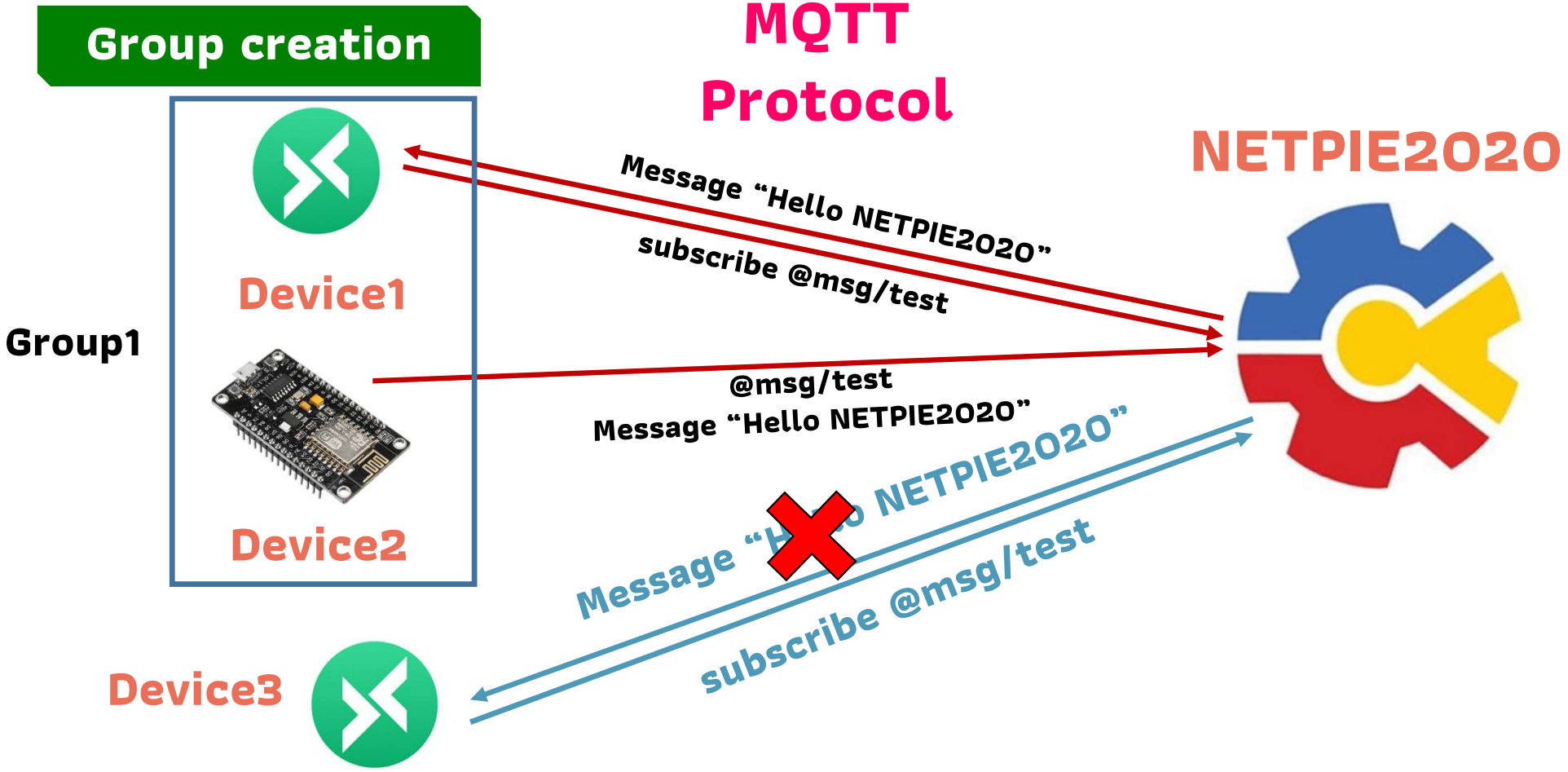
MQTT Protocol

NETPIE2020

NETPIE Portal

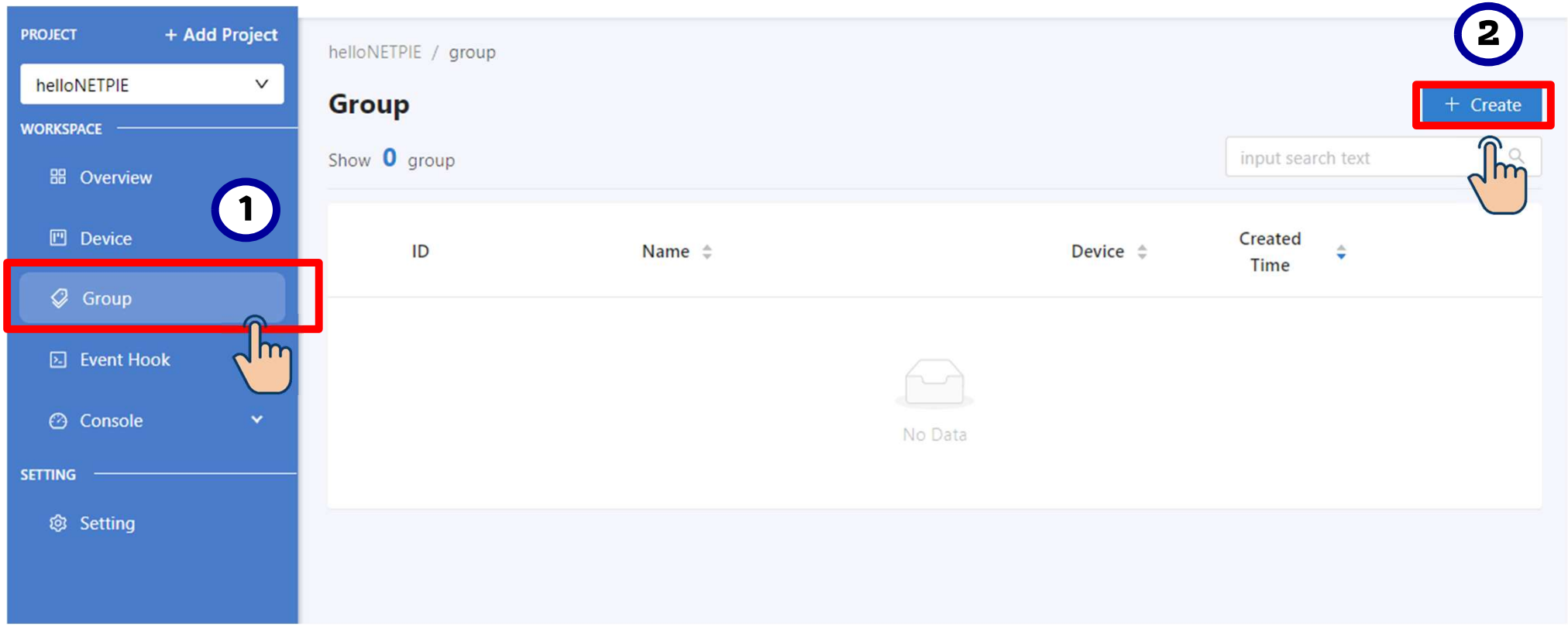


NETPIE Portal



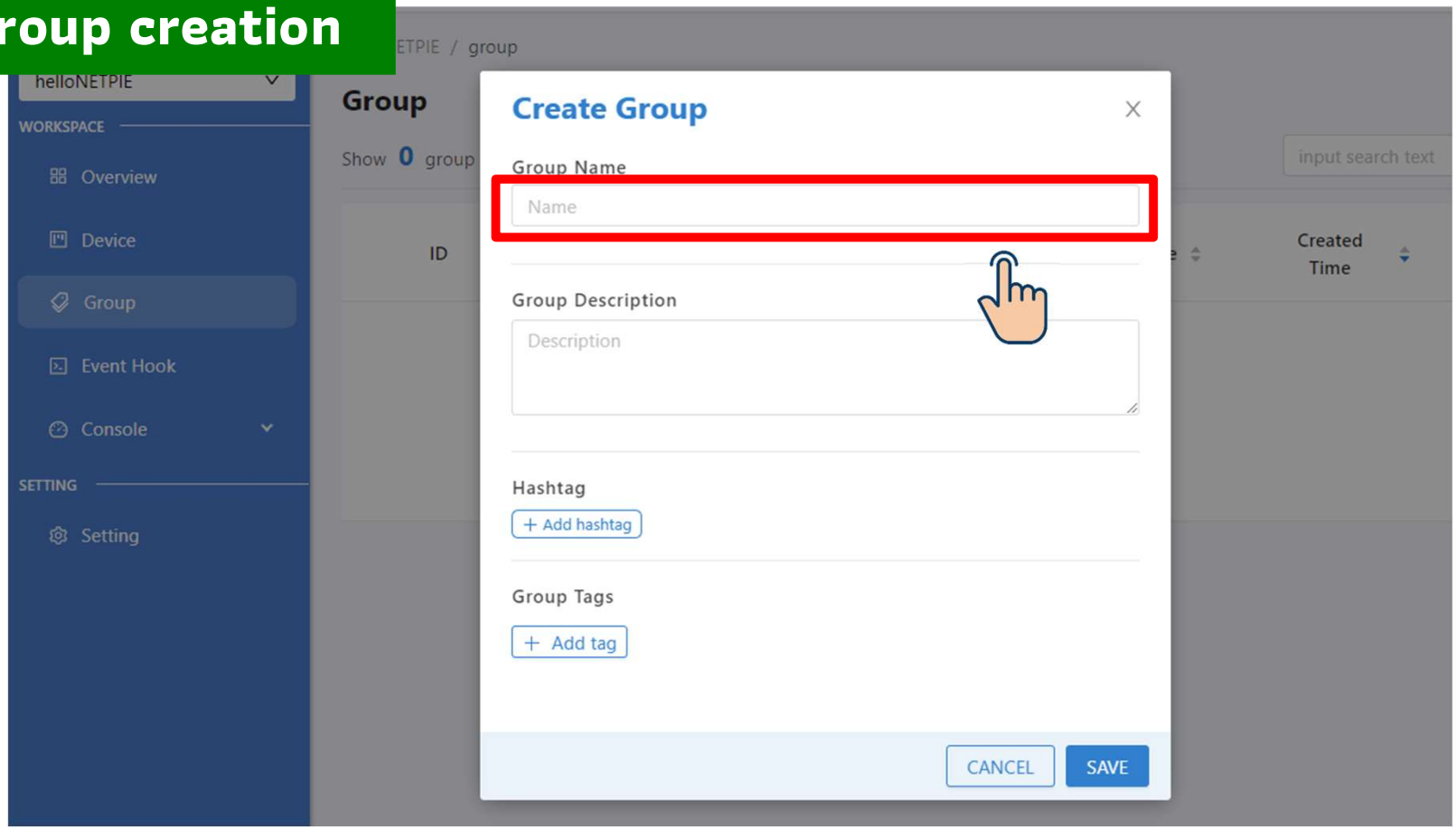
NETPIE Portal

Group creation



NETPIE Portal

Group creation

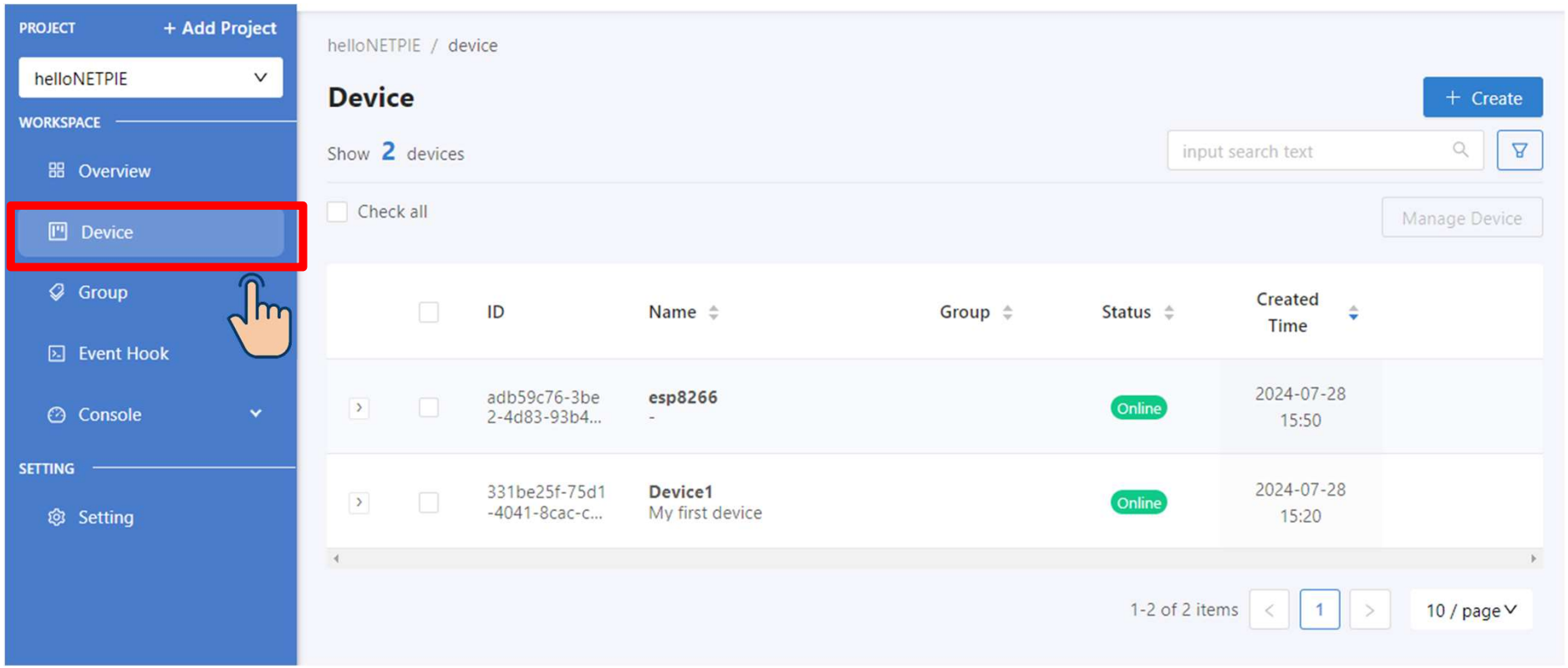


Group creation

Group is created with no device
An existing device must be brought into the group

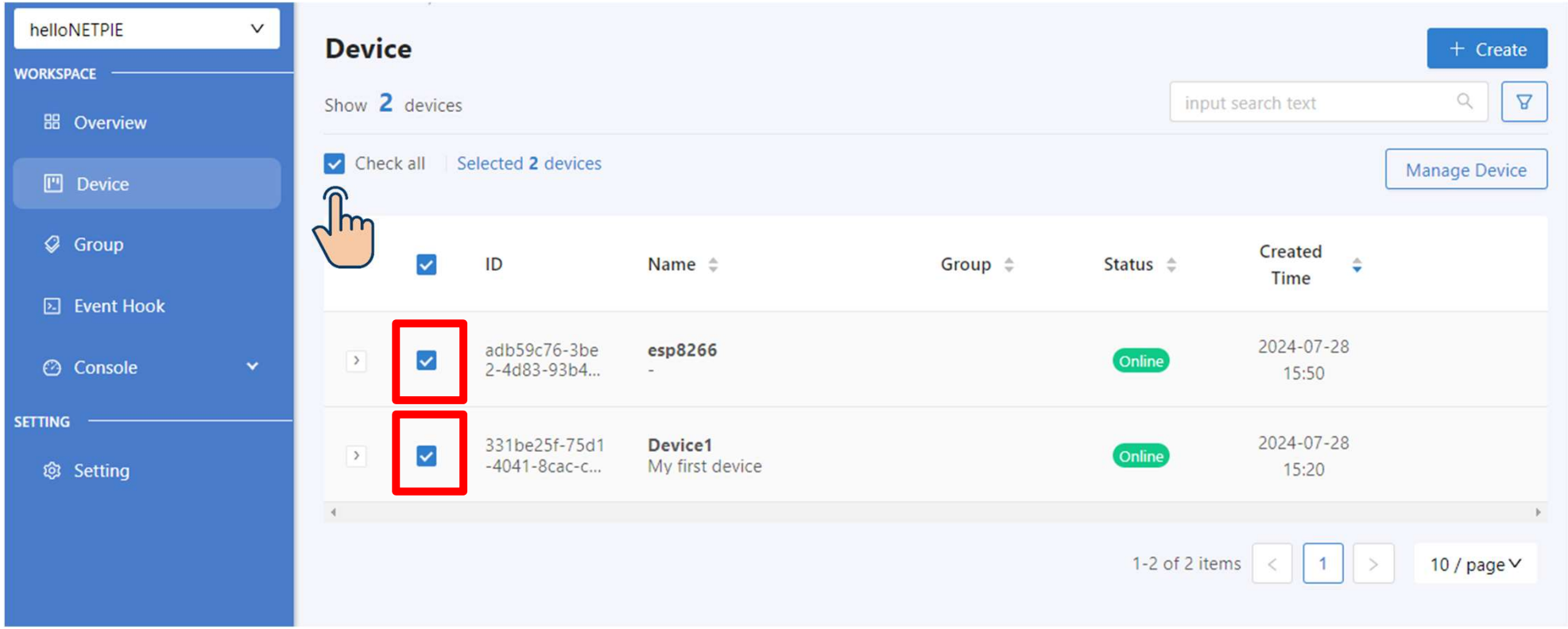
NETPIE Portal

Group creation



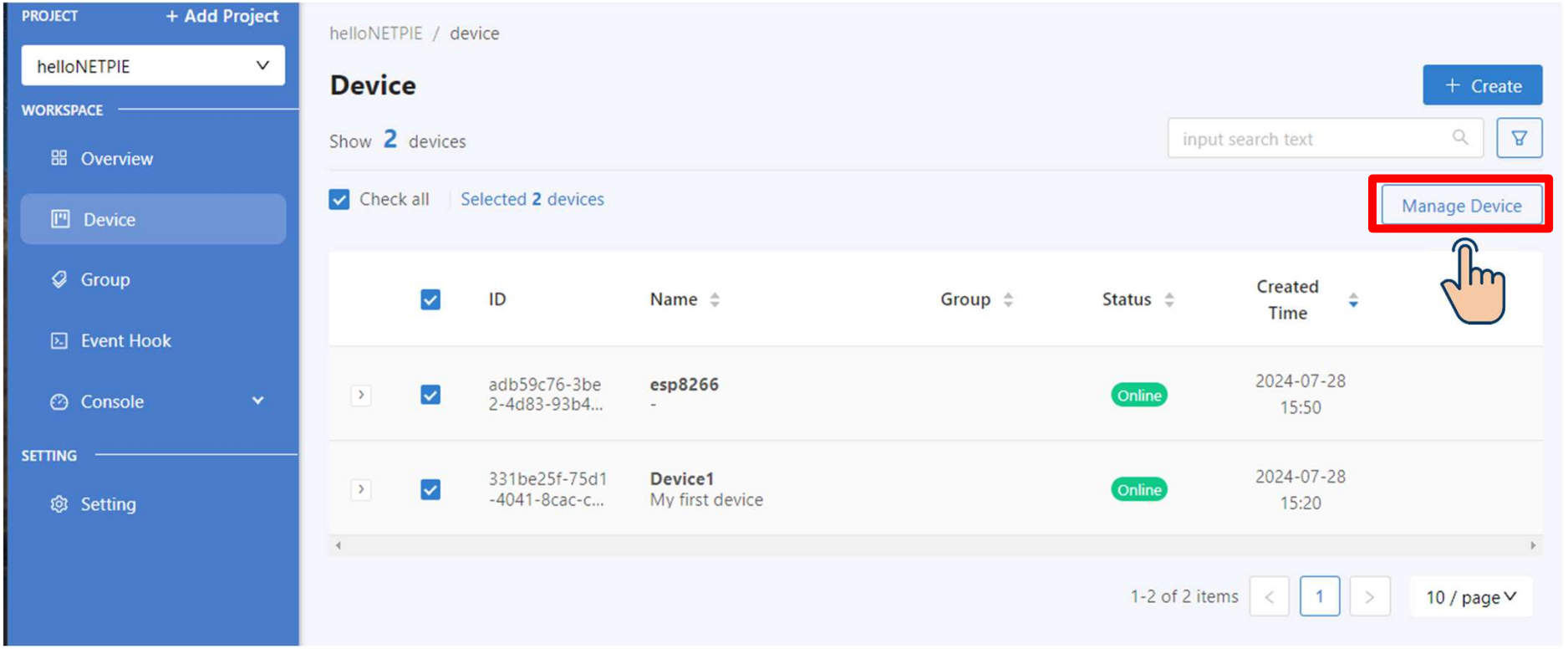
NETPIE Portal

Group creation



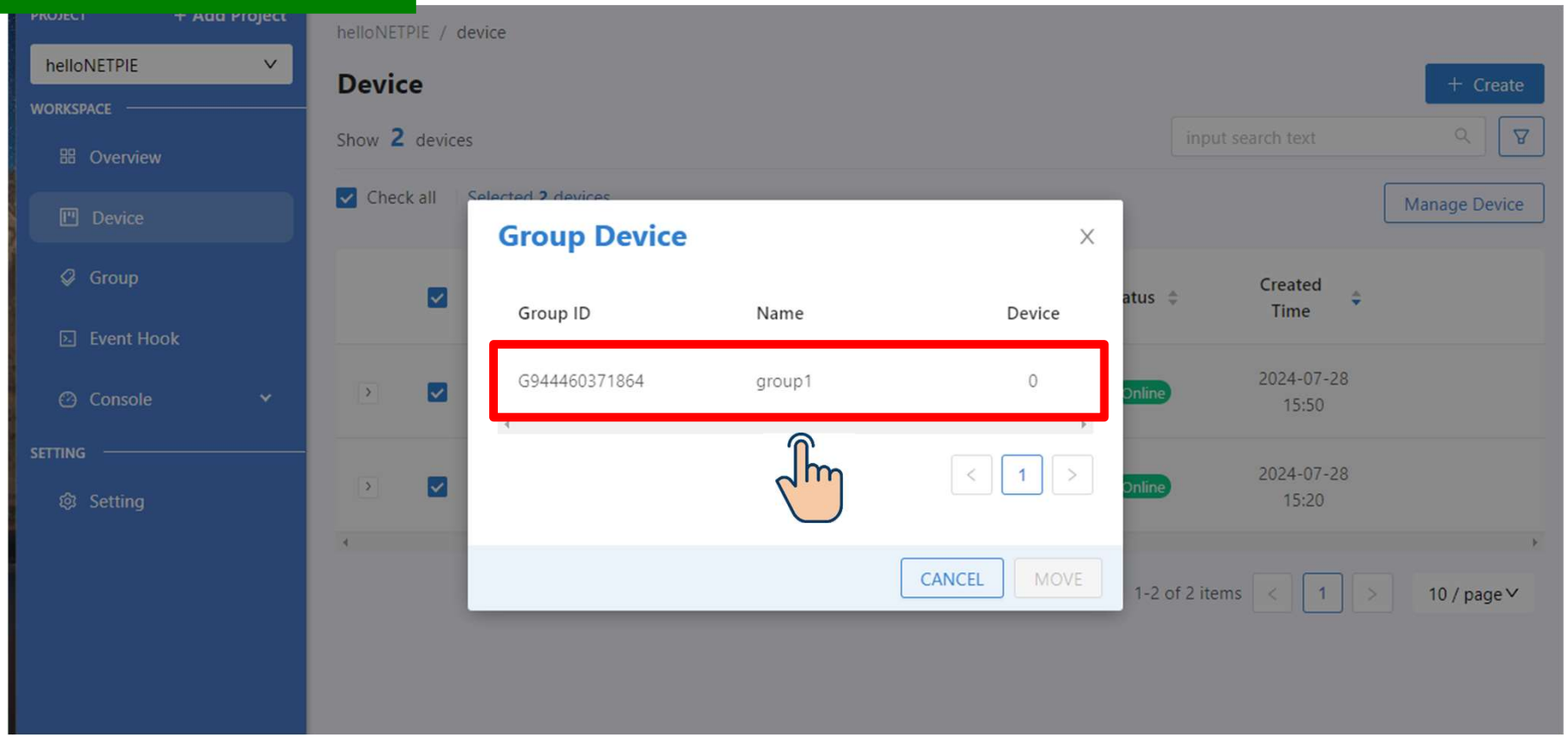
NETPIE Portal

Group creation



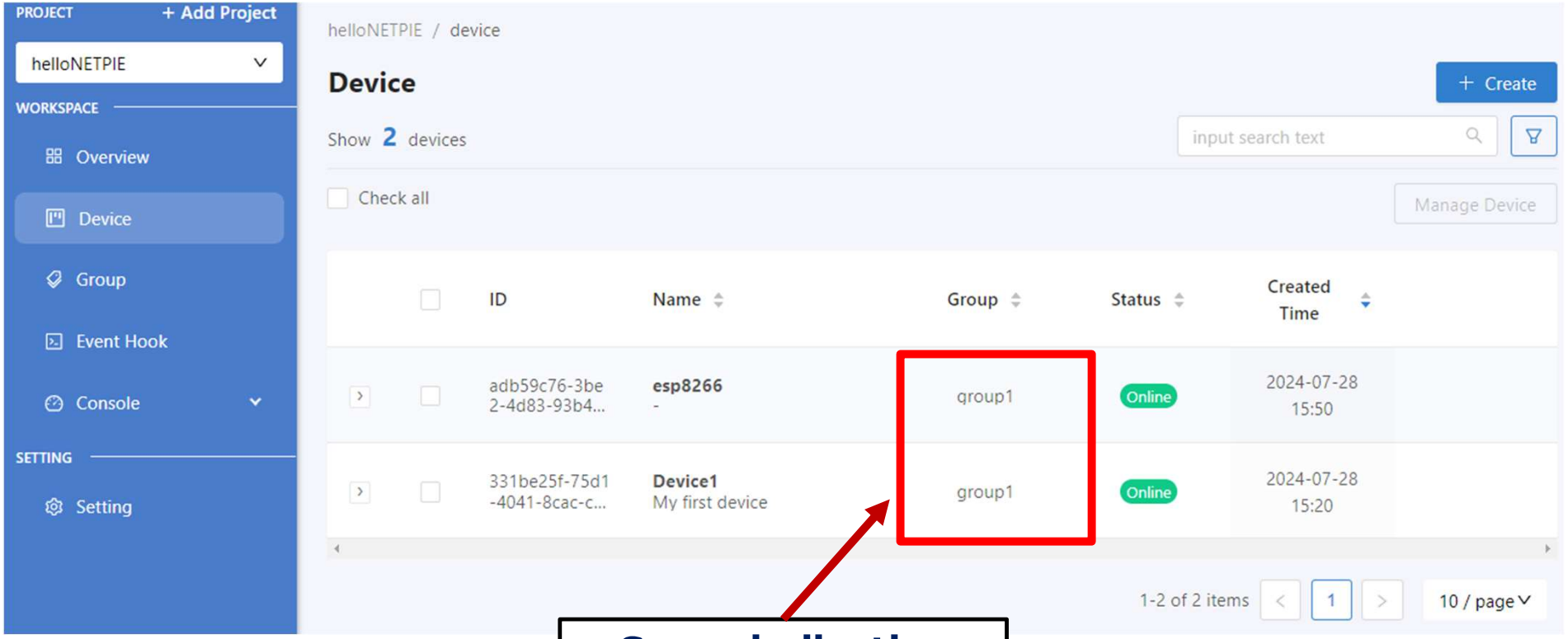
NETPIE Portal

Group creation



NETPIE Portal

Group creation



Group indication

NETPIE Portal

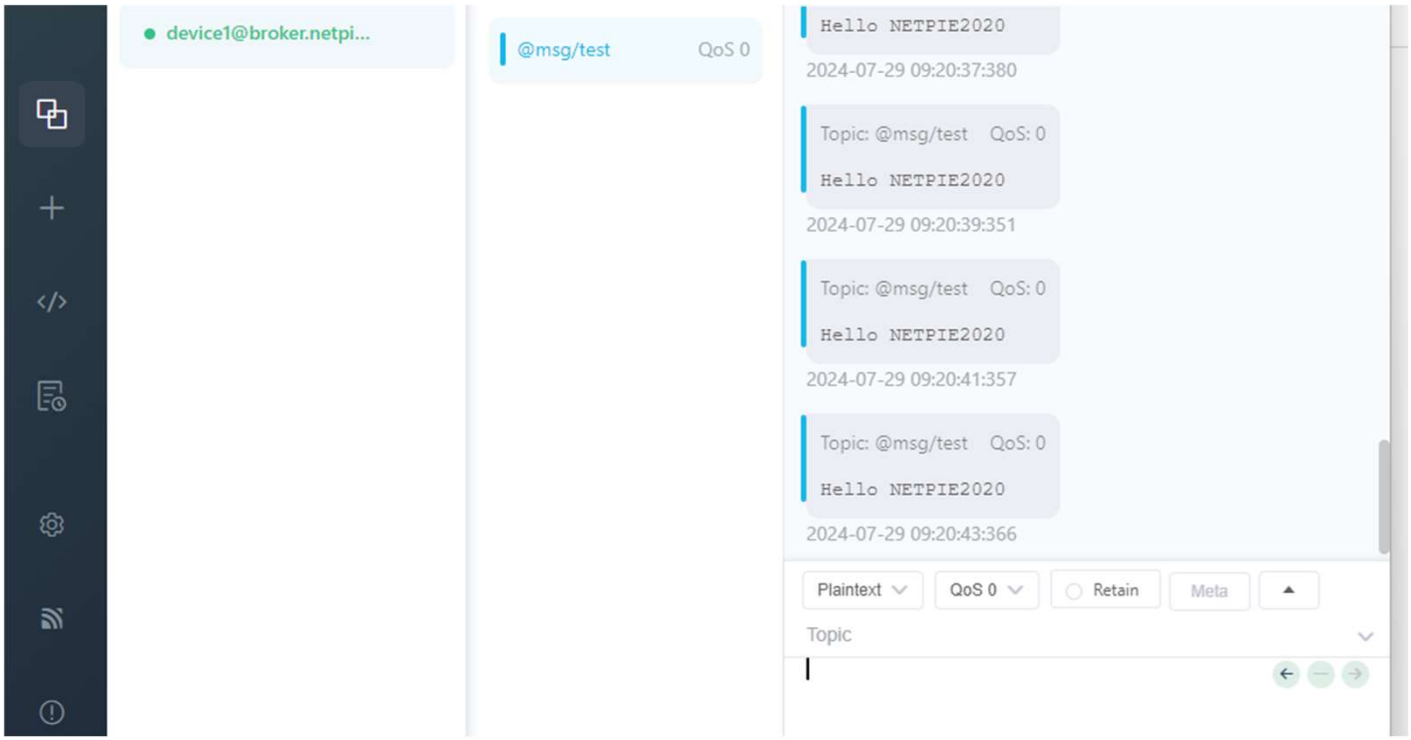
Group creation

More details can be observed in the Device Groups

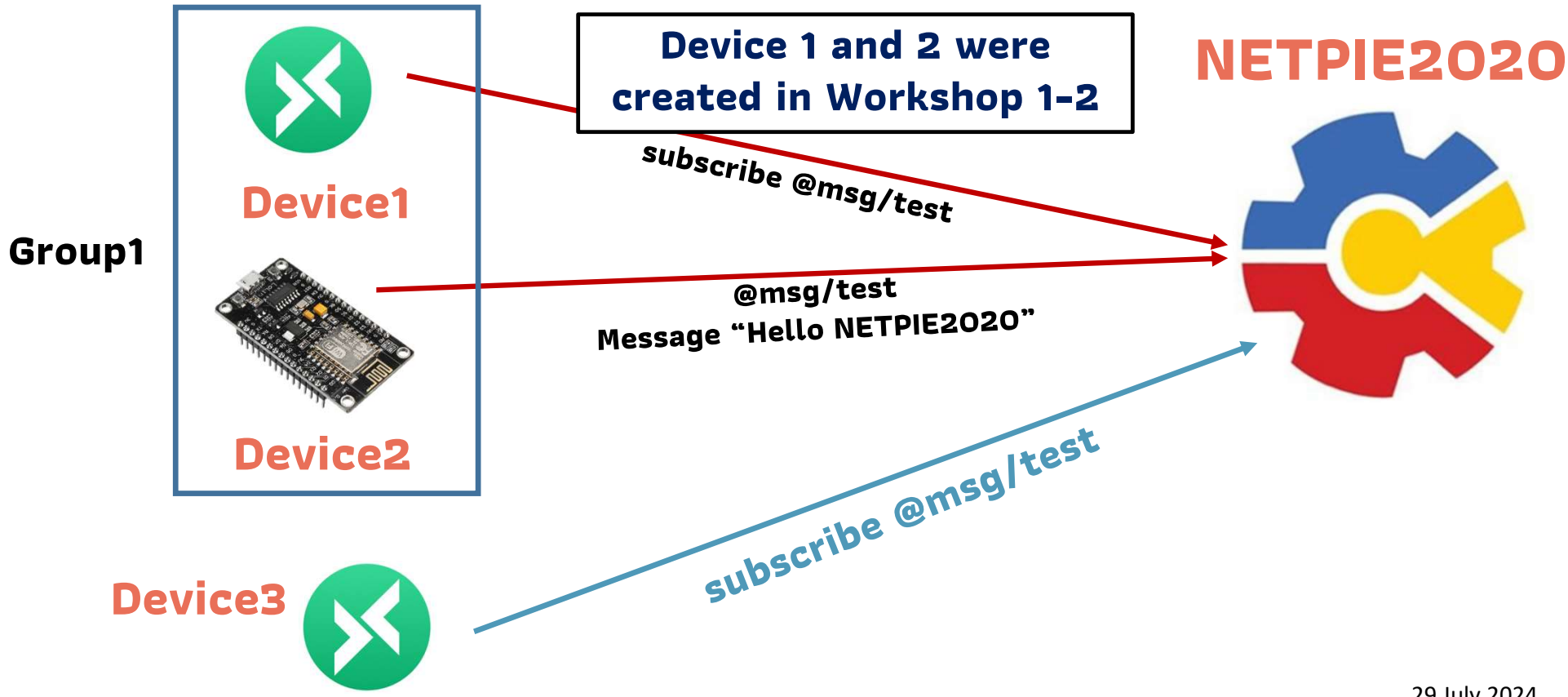
NETPIE Portal

Group creation

Messages from NodeMCU to MQTTX

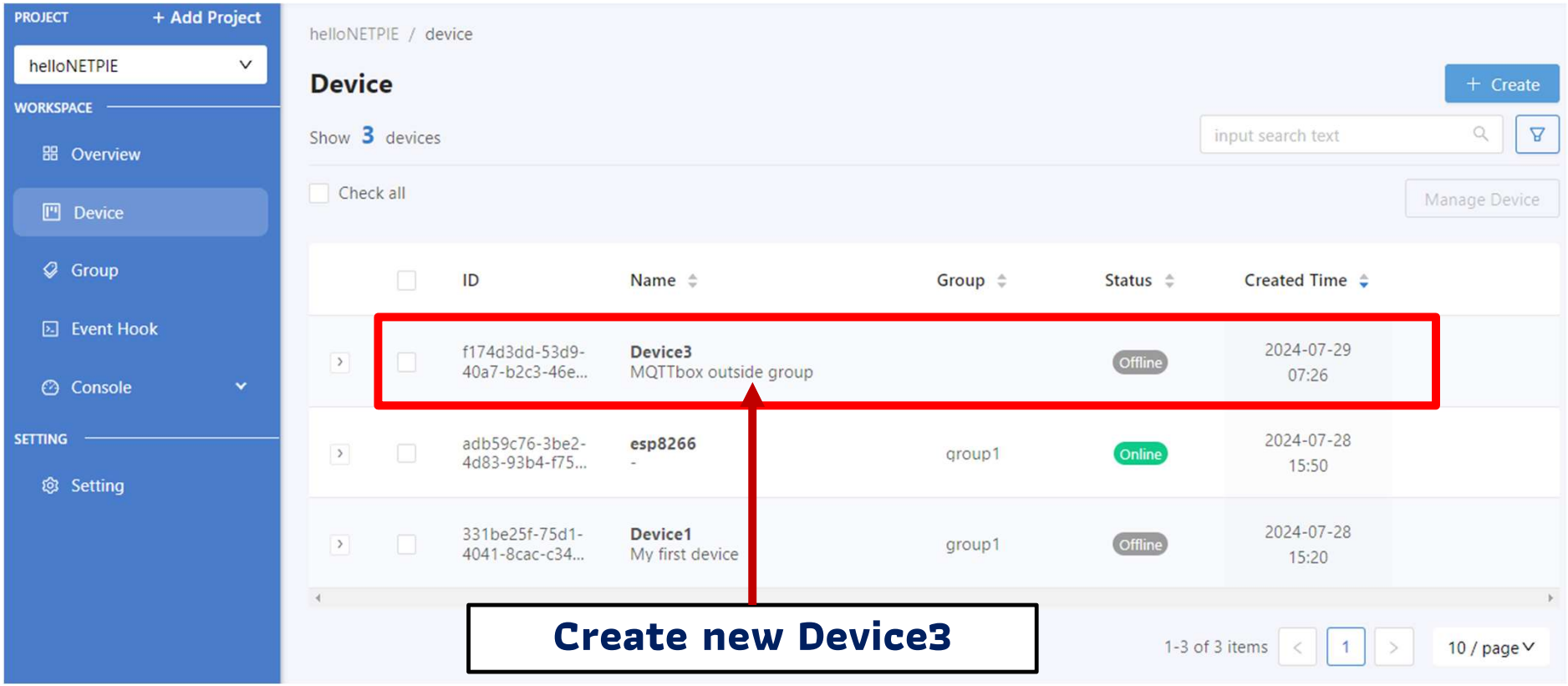


Workshop 3 : Communication within a Group and outside



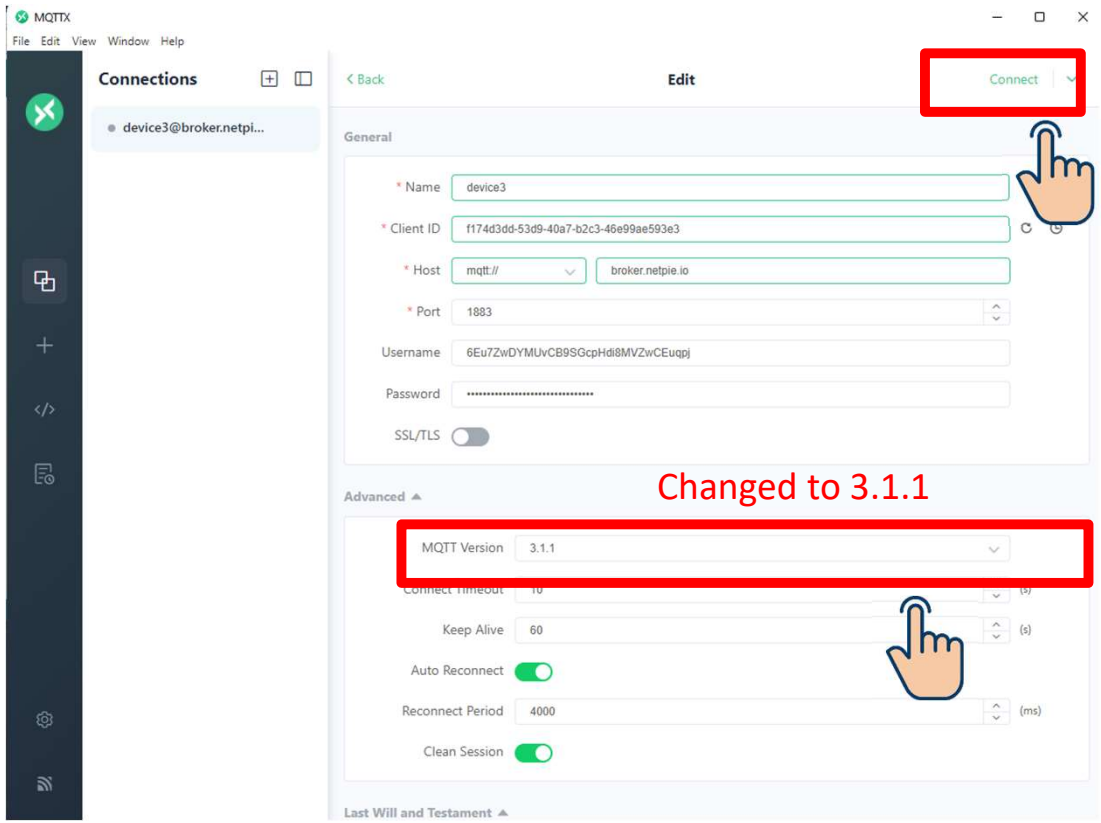
NETPIE Portal

Workshop 3 : Communication within a Group and outside

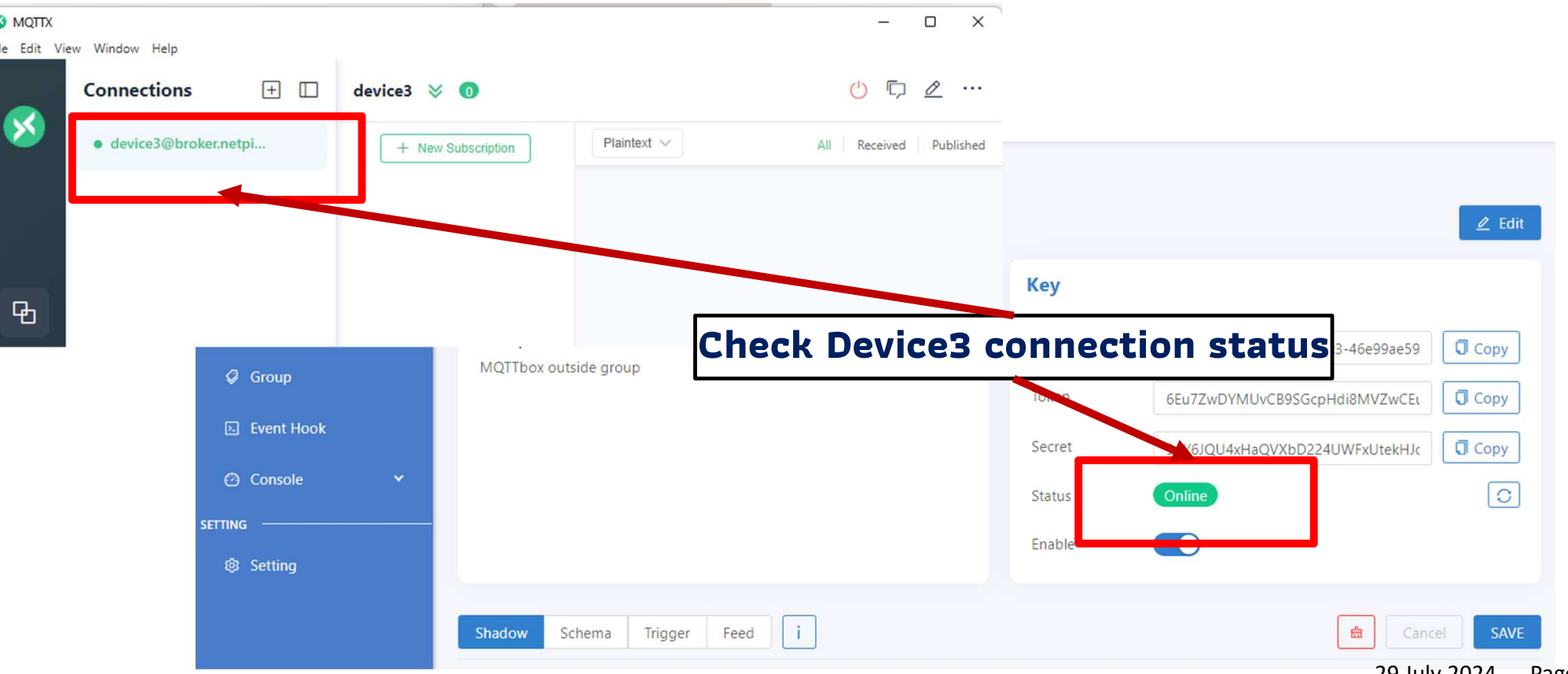


Workshop 3 : Communication within a Group and outside

Open new MQTTX window to connect with Device3

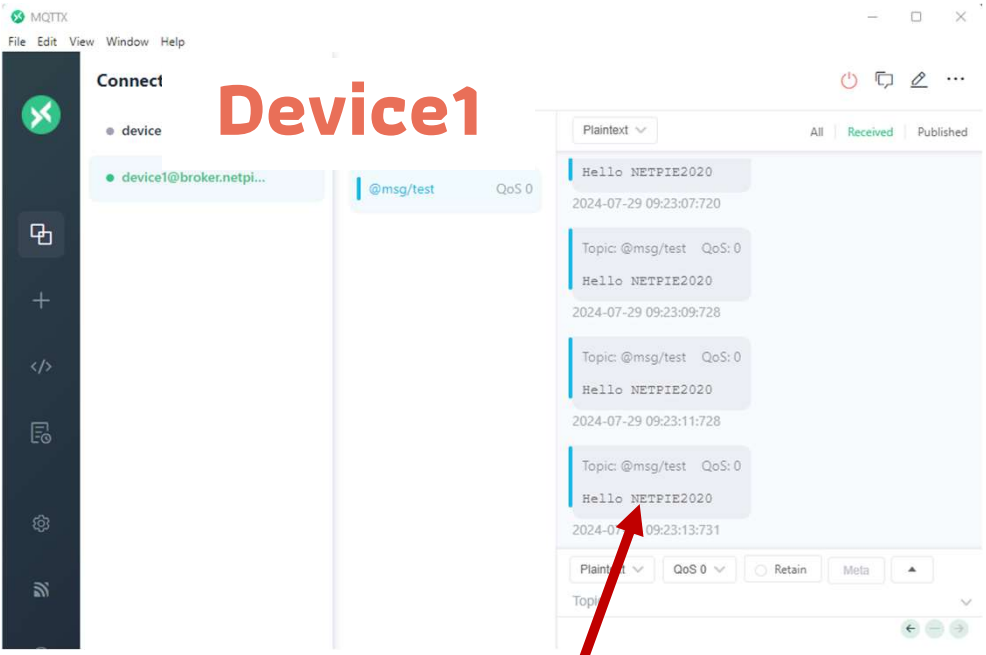


Workshop 3 : Communication within a Group and outside

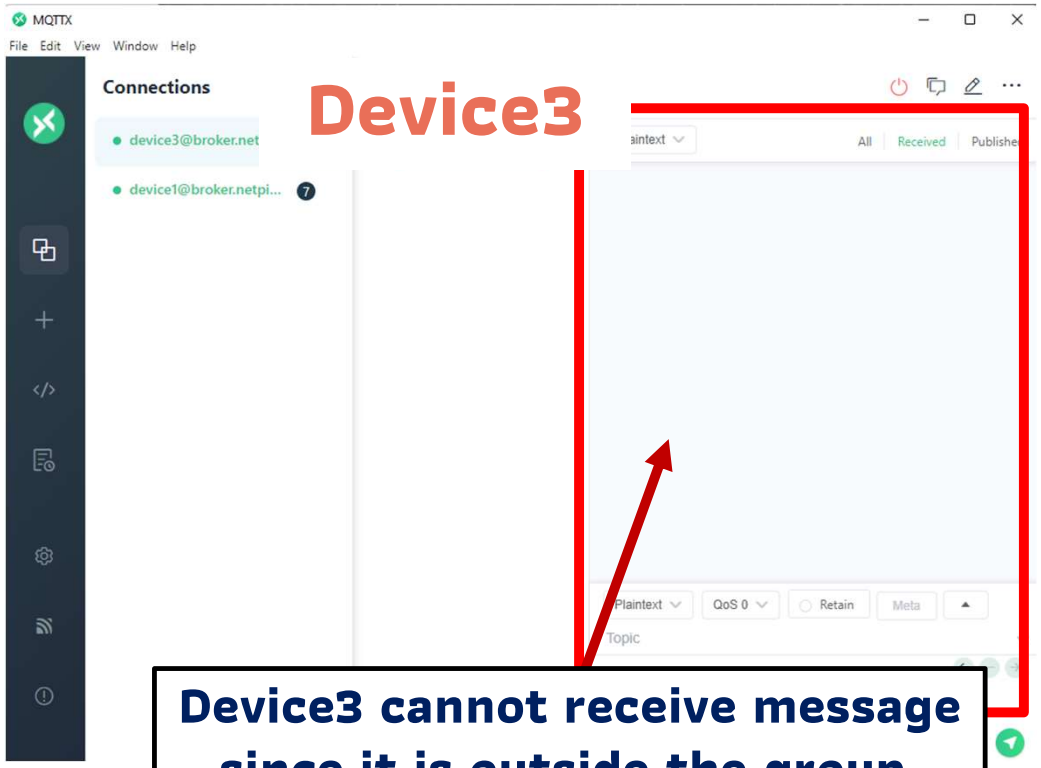


NETPIE Portal

Workshop 3 : Communication within a Group and outside



Device1



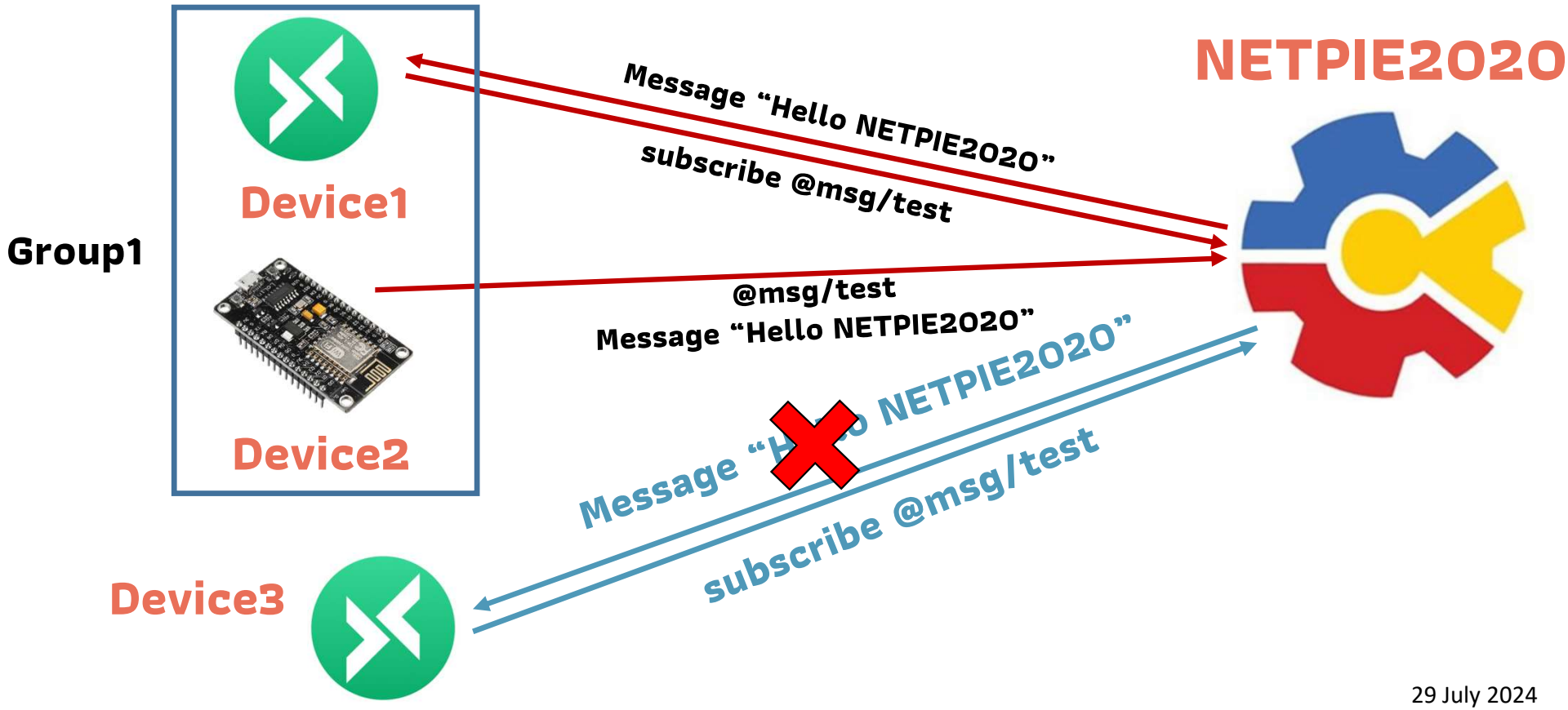
Device3

At Device1, it receives a message that NodeMCU sent, “Hello NETPIE2020”.

Device3 cannot receive message since it is outside the group.

NETPIE Portal

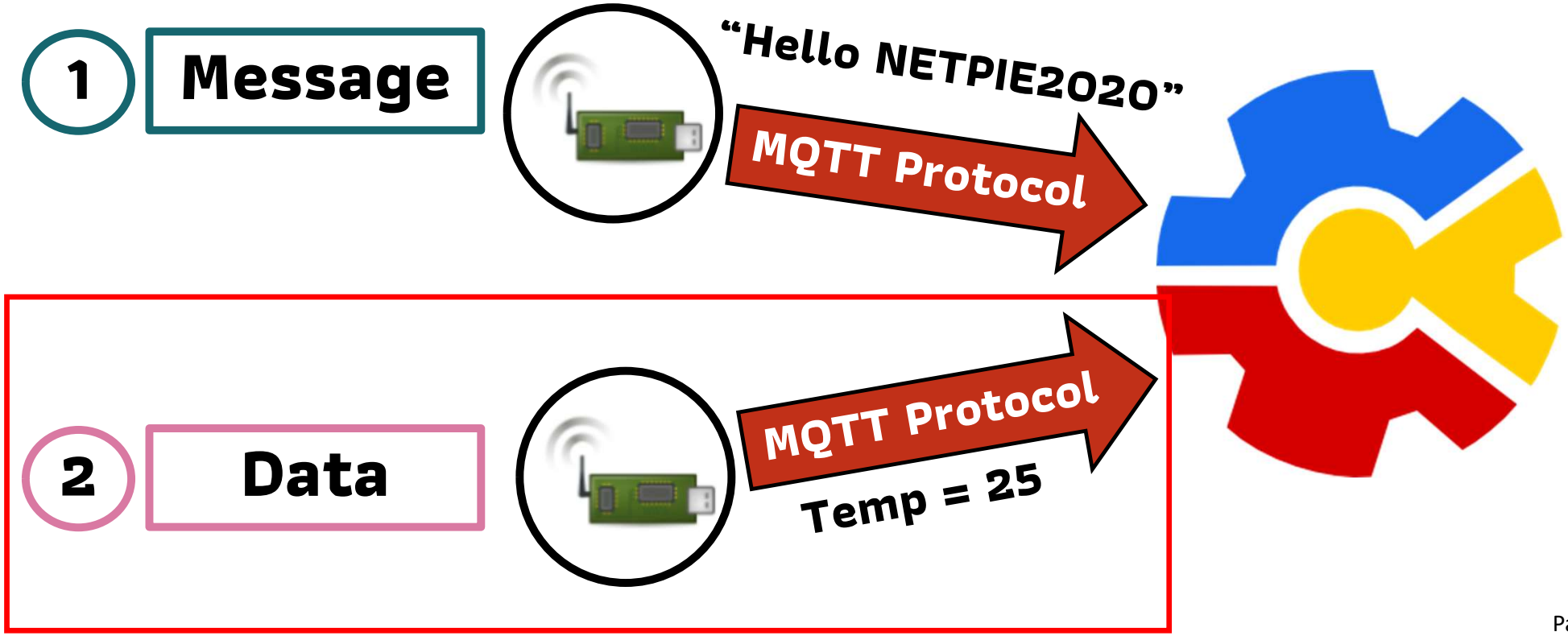
Workshop 3 : Communication within a Group and outside



Device Data Management

Device Data Management

There are two types of messages sent to NETPIE2020 via MQTT.



Device Data Management

NETPIE2020 manages device information in 4 main sections.

1. **Device Shadow : Latest device database**
2. **Device Schema : Device data structure**
3. **Device Trigger : Conditional device data**
4. **Event Hooks : Device transmission formats**

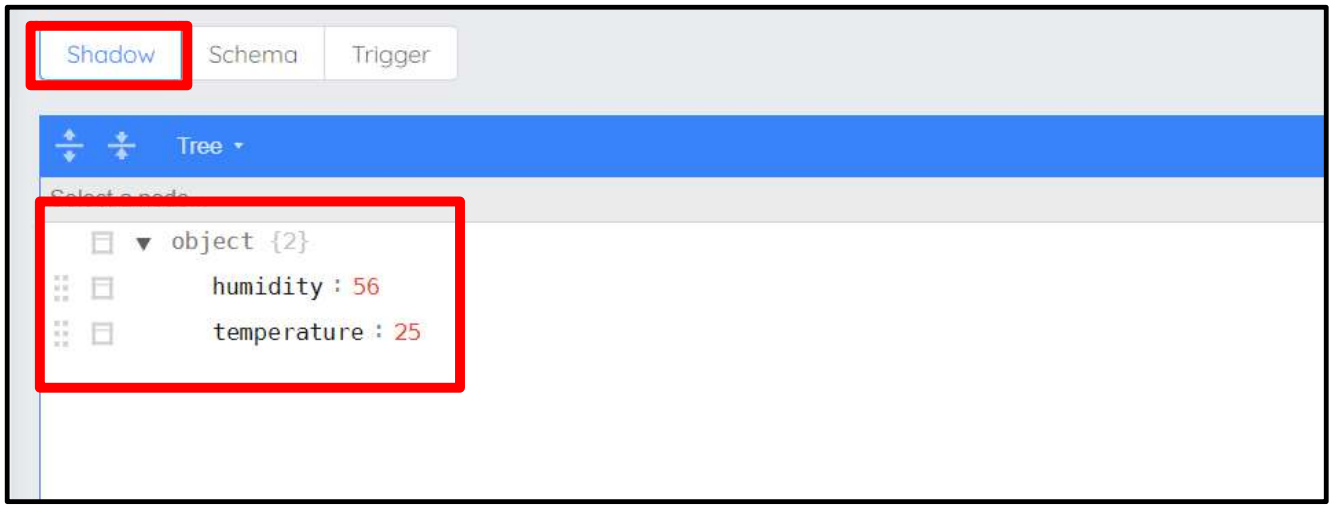
NETPIE2020

Device Data Management

Device Shadow

Device Shadow is the virtual database of the device that is allocated for every device for two types of data storage

1. **Device Shadow Data : sensor data, for example**
2. **Device Shadow State เช่น : device online/offline status, for example**



Note: previous Version appearance

Device Shadow

The MQTT topic that is involved with managing Device Shadow

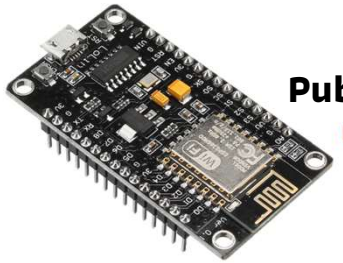
Shadow Topic

Used to manage own device shadow for publishing to edit Shadow information, and subscribing to receive Shadow information.

Publish Topic	Description	Subscribe Topic
@shadow/data/update	To update Shadow Data value by sending a payload in JSON format	**@shadow/data/updated**

Device Shadow

Example of sending data in JSON format



NodeMCU

@shadow/data/update

Publish : { “data”: { “Temp” : 24, “Humi” : 58 }}

NETPIE2020

“Temp” : 24

“Humi” : 58

Device Shadow

But in order to store data in Device Shadow, you need to create a Device Schema first.

Device Data Management

Device Schema

Device Schema is a data structure defined to manage Device Shadow. For devices that need data management, a Device Schema should be created. This Device Schema allows the server to

- **Check data types before storing**
- **Converting data before storage, such as changing data units.**
- **Data collection in Timeseries Database [Feed]**

Device Data Management

Device Schema

Declaring the Device Schema in JSON format.

```
{
 "additionalProperties": false,
 "properties": {
   "light": {
     "operation": {
       "store": {
         "ttl": "7d"
       }
     },
     "type": "number"
   },
   "temperature": {
     "operation": {
       "store": {
         "ttl": "7d"
       },
       "transform": {
         "expression": [{$.temperature}*1.8) + 32"
       }
     },
     "type": "number"
   }
 }
}
```

Device Schema consists of 2 main parts

additionalProperties

Properties

Device Data Management

Device Schema

Declaring the Device Schema in JSON format

```
{
 "additionalProperties": false,
 "properties": {
  "light": {
   "operation": {
    "store": {
     "ttl": "7d"
    }
   },
   "type": "number"
  },
  "temperature": {
   "operation": {
    "store": {
     "ttl": "7d"
    },
    "transform": {
     "expression": [($.temperature]*1.8) + 32"
    }
   },
   "type": "number"
  }
 }
}
```

additionalProperties

This is the permission to save data to Shadow or Timeseries Database in case the data is not declared in the properties section.

additionalProperties has 2 status values

true : authorize data write to Shadow or Timeseries Database

false : prohibit data write for data not defined in Properties

In the example, two items in properties are humidity and temperature

If the data received are temp, humid, light

addtionalProperties = true : will record temperature, humidity and light

addtionalProperties = false : will record only temperature, light

Device Data Management

Device Schema

Declaring the Device Schema in JSON format

```
{
 "additionalProperties": false,
"properties": {
  "light": {
   "operation": {
     "store": {
       "ttl": "7d"
     }
   },
   "type": "number"
  },
  "temperature": {
   "operation": {
     "store": {
       "ttl": "7d"
     },
     "transform": {
       "expression": [($.temperature]*1.8] + 32"
     }
   },
   "type": "number"
  }
 }
}
```

Properties

First, define a field name [for example, 'light' and 'Temperature'] and define properties for each field, which are divided into two parts:

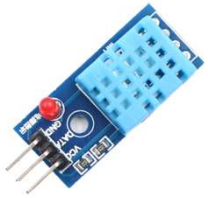
Operation For setting up data handling in that field, including
store for keeping data inTimeseries Database
ttl duration of data inTimeseries Database. Data that exceeds expiration date will be automatically deleted. To store system data, this value needs to be set in units of ms [milliseconds], s [seconds], m [minutes], h [hours], d [days], y [years]

Transform data transformation before keeping
expression formula for transformation
For example, a formula to convert Celsius to Fahrenheit= [$.temperature*1.8] + 32

Type data type in a field, such as number, string, array, object

Device Data Management

Workshop 4: Send Temperature and Light values from NodeMCU to NETPIE2020 and save them in Timeseries DB.



DHT11

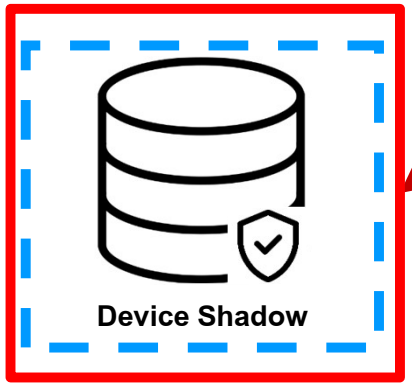
NodeMCU

@shadow/data/update

Publish : { “data”: { “Temp” : 24, “Humi” : 58, “place” : “NECTEC” }}

NETPIE2020

Note : if IGR board is used, light intensity is also written to shadow.

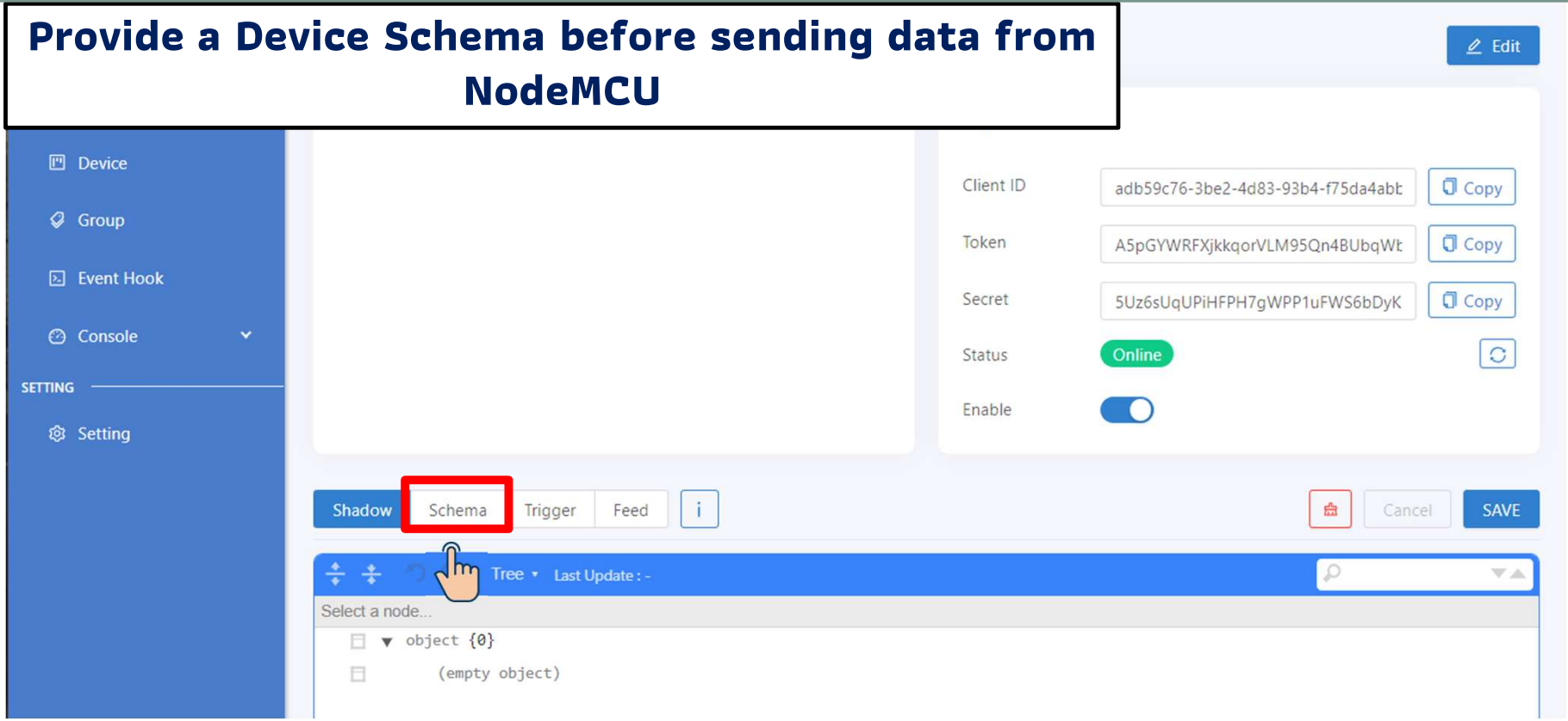


“Temp” : 75
“Humi” : 58
“place” : “NECTEC”

Device Data Management

Workshop 4: Send Temperature and Light values from NodeMCU to NETPIE2020 and save them in Timeseries DB.

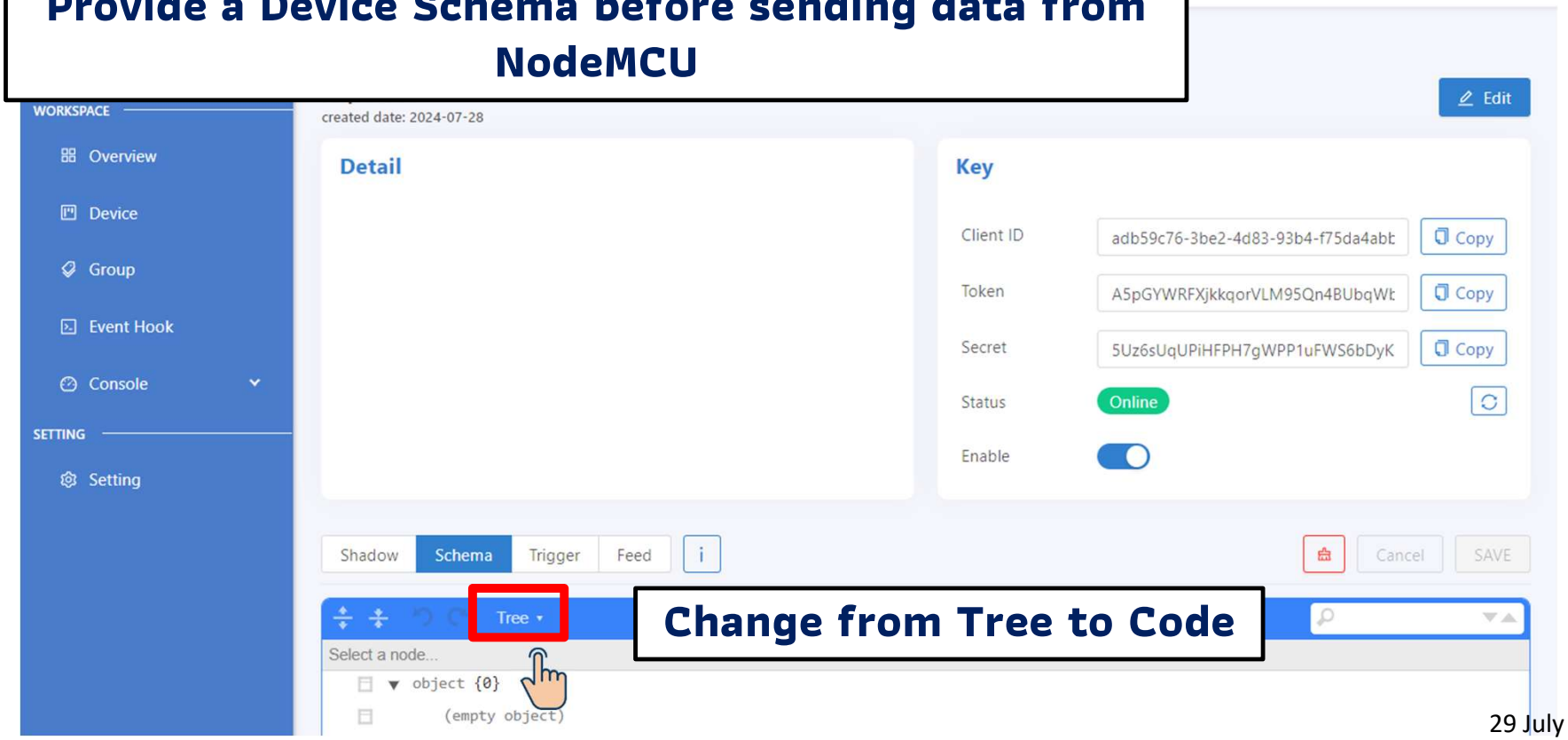
Provide a Device Schema before sending data from NodeMCU



Device Data Management

Workshop 4: Send Temperature and Light values from NodeMCU to NETPIE2020 and save them in Timeseries DB.

Provide a Device Schema before sending data from NodeMCU



Change from Tree to Code

Device Data Management

Workshop 4: Send Temperature and Light values from NodeMCU to NETPIE2020 and save them in Timeseries DB.

Device Schema in JSON format

```
{
 "additionalProperties": false,
 "properties": {
  "humidity": {
   "operation": {
    "store": {
     "ttl": "7d"
    }
   },
   "type": "number"
  },
  "temperature": {
   "operation": {
    "store": {
     "ttl": "7d"
    },
    "transform": {
     "expression": "[{$.temperature]*1.8] + 32"
    }
   },
   "type": "number"
  },
  "place": {
   "operation": {
    "store": {
     "ttl": "7d"
    }
   },
   "type": "string"
  }
 }
}
```

Copy to Device Schema

Device Data Management

Workshop 4: Send Temperature and Light values from NodeMCU to NETPIE2020 and save them in Timeseries DB.

Device Data Management

Workshop 4: Send Temperature and Light values from NodeMCU to NETPIE2020 and save them in Timeseries DB.

Device Schema in JSON format

Set **additionalProperties = false**
In order not to save values other than listed in properties to Shadow.

The first variable is **light**
with properties
- Store for 7 days
- Data type is number

```
{
"additionalProperties": false,
"properties": {
  "humidity": {
    "operation": {
      "store": {
        "ttl": "7d"
      }
    },
    "type": "number"
  },
  "temperature": {
    "operation": {
      "store": {
        "ttl": "7d"
      },
      "transform": {
        "expression": "[[$.temperature]*1.8] + 32"
      }
    },
    "type": "number"
  },
  "place": {
    "operation": {
      "store": {
        "ttl": "7d"
      }
    },
    "type": "string"
  }
 }
}
```

Device Data Management

Workshop 4: Send Temperature and Light values from NodeMCU to NETPIE2020 and save them in Timeseries DB.

Device Schema in JSON format

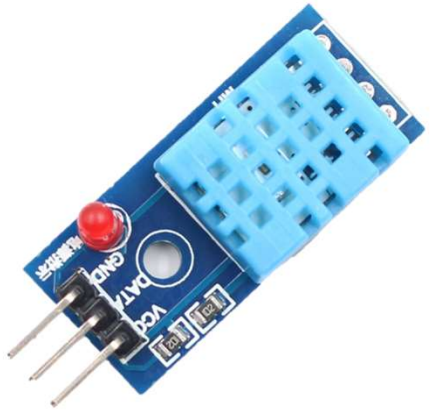
```
{
 "additionalProperties": false,
 "properties": {
  "humidity": {
   "operation": {
    "store": {
     "ttl": "7d
    }
   },
   "type": "number"
  },
  "temperature": {
   "operation": {
    "store": {
     "ttl": "7d"
    },
    "transform": {
     "expression": "[{$.temperature]*1.8] + 32"
    }
   },
   "type": "number"
  },
  "place": {
   "operation": {
    "store": {
     "ttl": "7d"
    }
   },
   "type": "string"
  }
 }
}
```

Variable to store is **temperature** with properties
- Store for 7 days
- Convert from Celsius to Fahrenheit
- Variable type is number

Variable to store is **place** with properties
- Store for 7 days
- Variable type is string

Workshop 4: Send Temperature and Light values from NodeMCU to NETPIE2020 and save them in Timeseries DB.

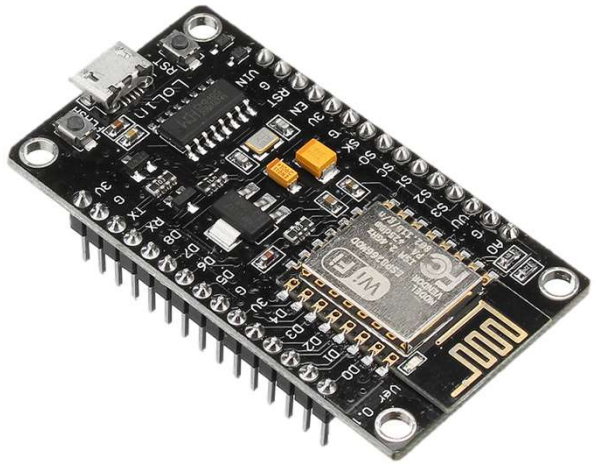
Connect DHT11 to NodeMCU



Data -> D4

Vcc -> 3v3

GND -> GND



DHT11

NodeMCU

Workshop 4: Send Temperature and Light values from NodeMCU to NETPIE2020 and save them in Timeseries DB.

Open Workshop4.ino

Code in Workshop4.ino consists of 3 parts

1. **Part 1 Library and variable declaration**
   - Include required libraries
   - Declare pins for DHT11
   - Variable declaration and assignment to connect to WiFi and MQTT
   - Variable declarations
   - MQTT commands

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include "DHT.h"

#define DHTPIN D4
#define DHTTYPE DHT11

const char* ssid = "Your_SSID";
const char* password = "Your_Password";
const char* mqtt_server = "broker.netpie.io";
const int mqtt_port = 1883;
const char* mqtt_Client = "Client_ID";
const char* mqtt_username = "Token";
const char* mqtt_password = "Secret";

WiFiClient espClient;
PubSubClient client(espClient);
DHT dht(DHTPIN, DHTTYPE);

char msg[100]
```

Workshop 4: Send Temperature and Light values from NodeMCU to NETPIE2020 and save them in Timeseries DB.

2 **Part 2 : various functions**

MQTT connection function

```
void reconnect() {
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    if (client.connect(mqtt_Client, mqtt_username, mqtt_password)) {
      Serial.println("connected");
    }
    else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println("try again in 5 seconds");
      delay(5000);
    }
  }
}
```

MQTT connection
- “connected” is shown when succeed
- if not, “failed …” is shown and reconnect is attemped automatically

Workshop 4: Send Temperature and Light values from NodeMCU to NETPIE2020 and save them in Timeseries DB.

2 Part 2 : various functions

```
void setup() {
  Serial.begin(115200);
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
  client.setServer(mqtt_server, mqtt_port);
  dht.begin();
}
```

Setup function

Connect WiFi and MQTT with specified settings

Start the DHT object instance

Workshop 4: Send Temperature and Light values from NodeMCU to NETPIE2020 and save them in Timeseries DB.

3 **Part 3 : loop() function**

Read humidity and temperature data from sensor, declare the place before sending to NETPIE2020

```
void loop() {
  int humidity = dht.readHumidity();
  int temperature = dht.readTemperature();
  String place = "NECTEC Thailand";
   if (!client.connected()) {
   reconnect();
  }
  client.loop();
  String data = "{\"data\": {\"humidity\":" + String(humidity) + ", \"temperature\":" +
String(temperature) + ", \"place\":" +String(place) + "}}";
  Serial.println(data);
  data.toCharArray(msg, (data.length() + 1));
  client.publish("@shadow/data/update", msg);
  delay(2000);
}
```

To maintain the connection state of MQTT

command to send message toTopic : @shadow/data/update every 2 seconds

Workshop 4: Send Temperature and Light values from NodeMCU to NETPIE2020 and save them in Timeseries DB.

Sending message in JSON format on Arduino IDE

Message to be sent is

{ "data" : { "humidity" : 52, "temperature" : 25, "place" : "NECTEC Thailand" }}

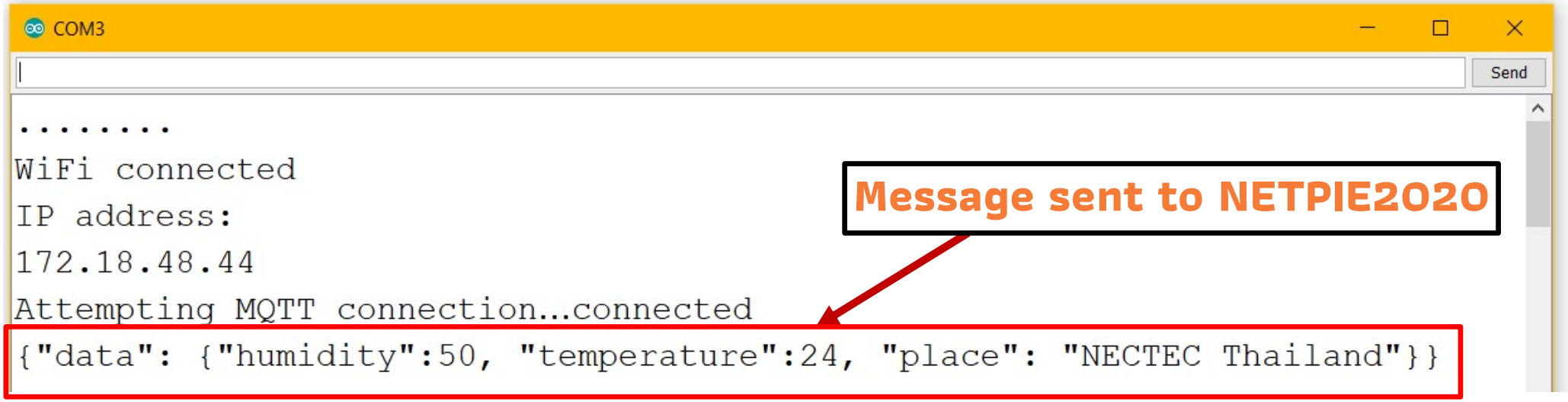
But the compiler on Arduino IDE sees " as String declaration. To fix this, put \ in front of " so that the program interprets " as part of the message

```
String data = "{\"data\": {\"humidity\":" + String(humidity) + ",
                \"temperature\":" + String(temperature) + ",
                \"place\":" +String(place) + "}}";
```

Note : Light intensity is added for IGR board

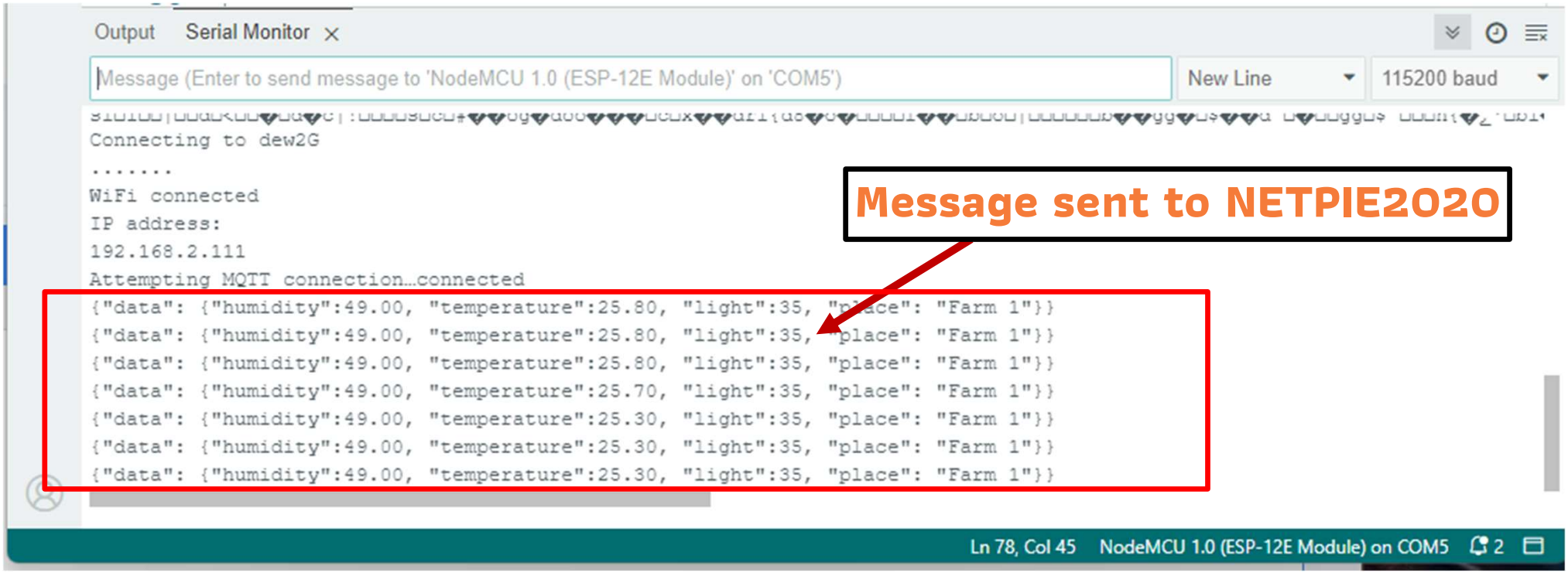
Device Data Management

Workshop 4: Send Humidity and Temperature values from NodeMCU to NETPIE2020 and save them in Timeseries DB.



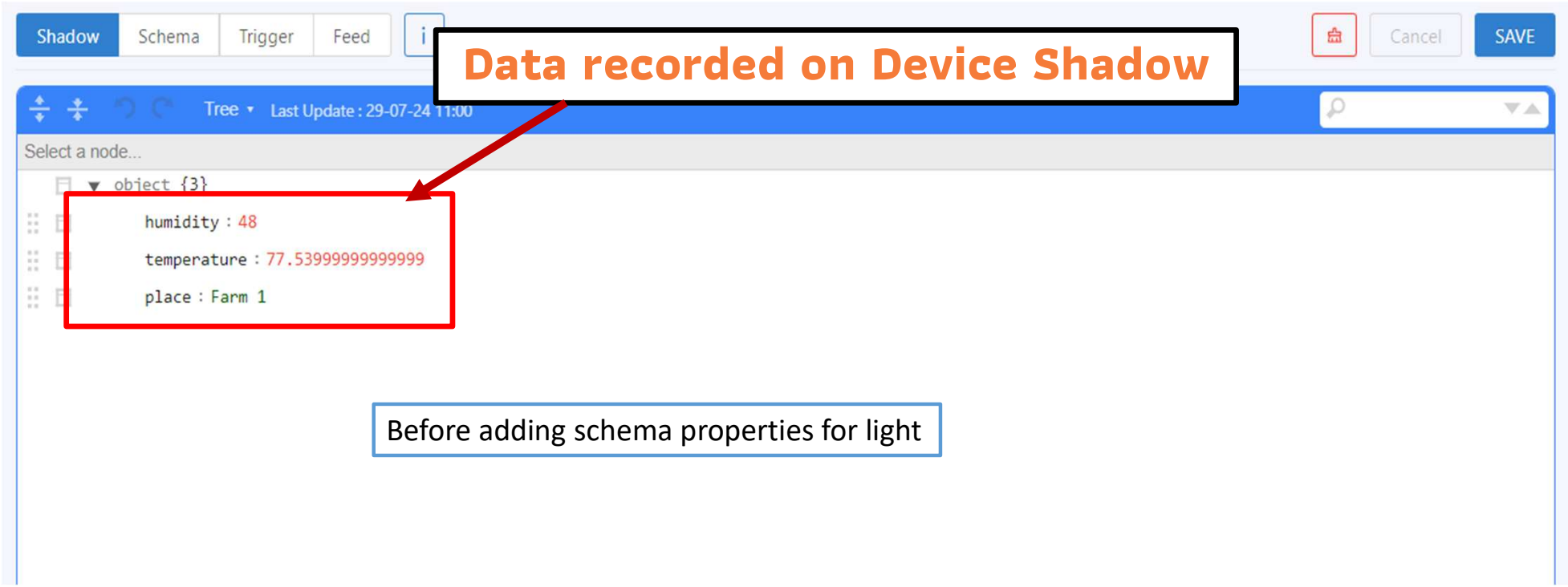
Device Data Management

Workshop 4: Send Humidity and Temperature and Light values from NodeMCU to NETPIE2020 and save them in Timeseries DB (IGR board)



Device Data Management

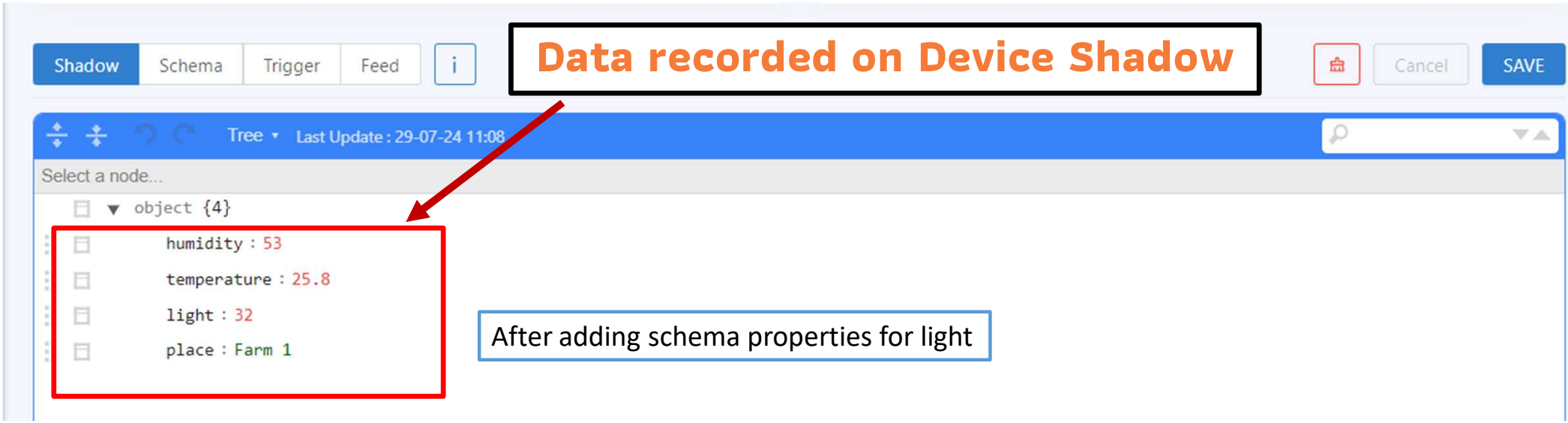
Workshop 4: Send sendor values from NodeMCU to NETPIE2020 and save them in Timeseries DB.



Before adding schema properties for light

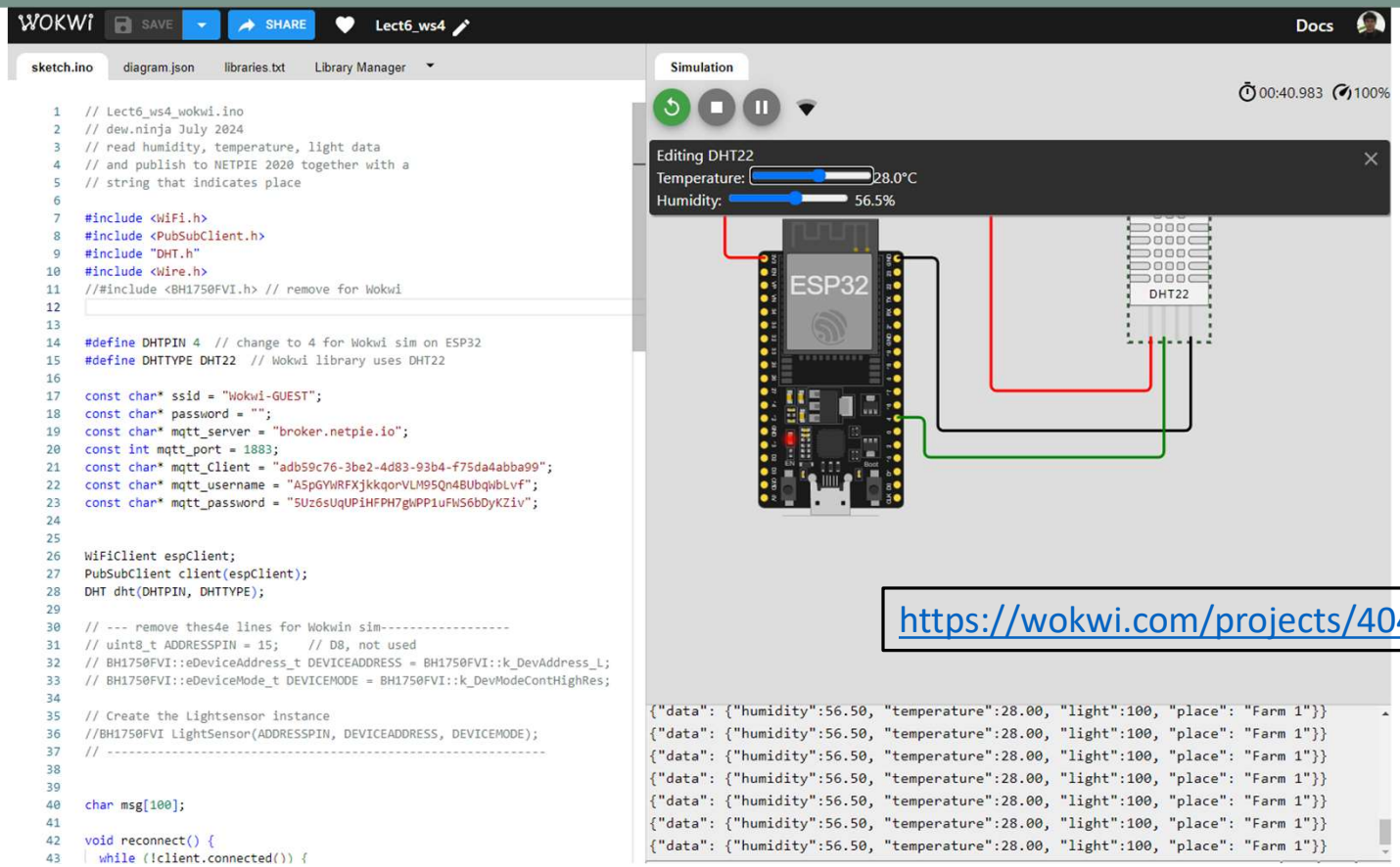
Device Data Management

Workshop 4: Send sendor values from NodeMCU to NETPIE2020 and save them in Timeseries DB.



Device Data Management

Workshop 4: Wokwi simulation



https://wokwi.com/projects/404716661370399745

Note for Device Trigger and Event Hook slides

Since this topic is not covered in my course. The original slides are left intact.

Device Trigger and Event Hook

It is a system that binds to changing Device Shadow information to external actions (Event Hook), such as setting alerts according to different status. According to the condition of the device set by the Trigger will be declared in JSON format:

```
{
   "enabled": true,
   "trigger": [
      {
         "action": "EVENT_HOOK_NAME",
         "event": "SHADOW.UPDATED or DEVICE.STATUSCHANGED",
         "condition": "Operation List ==, !=, >, >=, <, <=, in",
         "msg": "text",
         "option": {}
      }
   ]
}
```

Device Trigger and Event Hook

Trigger Format consists of 2 parts

1. enable is used to turn the Trigger on/off

```
{
  "enabled": true,
  "trigger": [
     {
        "action": "EVENT_HOOK_NAME",
        "event": "SHADOW.UPDATED or DEVICE.STATUSCHANGED",
        "condition": "Operation List ==, !=, >, >=, <, <=, in",
        "msg": "text",
        "option": {}
     }
  ]
}
```

Device Data Management

Device Trigger and Event Hook

- action : what Trigger has to do when an event occurs. Specify Event Hook name.

Trigger Format consist of 2 parts

2. trigger is for Trigger settings

```
{
  "enabled": true,
  "trigger": [
    {
      "action": "EVENT_HOOK_NAME",
      "event": "SHADOW.UPDATED or DEVICE.STATUSCHANGED",
      "condition": "Operation List ==, !=, >, >=, <, <=, in",
      "msg": "text",
      "option": {}
    }
  ]
}
```

Device Shadow

SHADOW.UPDATED : happens when Device Shadow Data changes according to certain condition (in this case we have to specify some condition, otherwise the Trigger would not fire)

DEVICE.STATUSCHANGED : happens when the Device changes its platform connection status from Or

Device Trigger and Event Hook

Trigger Format consist of 2 parts

```
{
   "enabled": true,
   "trigger": [
      {
         "action": "EVENT_HOOK_NAME",
         "event": "SHADOW.UPDATED or DEVICE.STATUSCHANGED",
         "condition": "Operation List ==, !=, >, >=, <, <=, in",
         "msg": "text",
         "option": {}
      }
   ]
}
```

Condition : conditional change in Device Shadow Data . Use in cas of SHADOW.UPDATED

msg : a message to notify the user when Trigger fires

option : use for specify some other parameters

For reference, variables in Trigger can be found in https://docs.nexpie.io/device-config.html#device-trigger-and-event-hook

Device Trigger and Event Hook

Trigger and Event Hook example

Event Hooks
LINE NOTIFY

Device2

Publish : @shadow/data/update

NETPIE2020

Device2 Shadow

"Temp" : 75
"Humi" : 58
Status : Online

Trigger : DEVICE.STATUSCHANGE

Device Data Management

Device Trigger and Event Hook

Trigger and Event Hook example

Device2

Publish : @shadow/data/update

Event Hooks
LINE NOTIFY

NETPIE2020

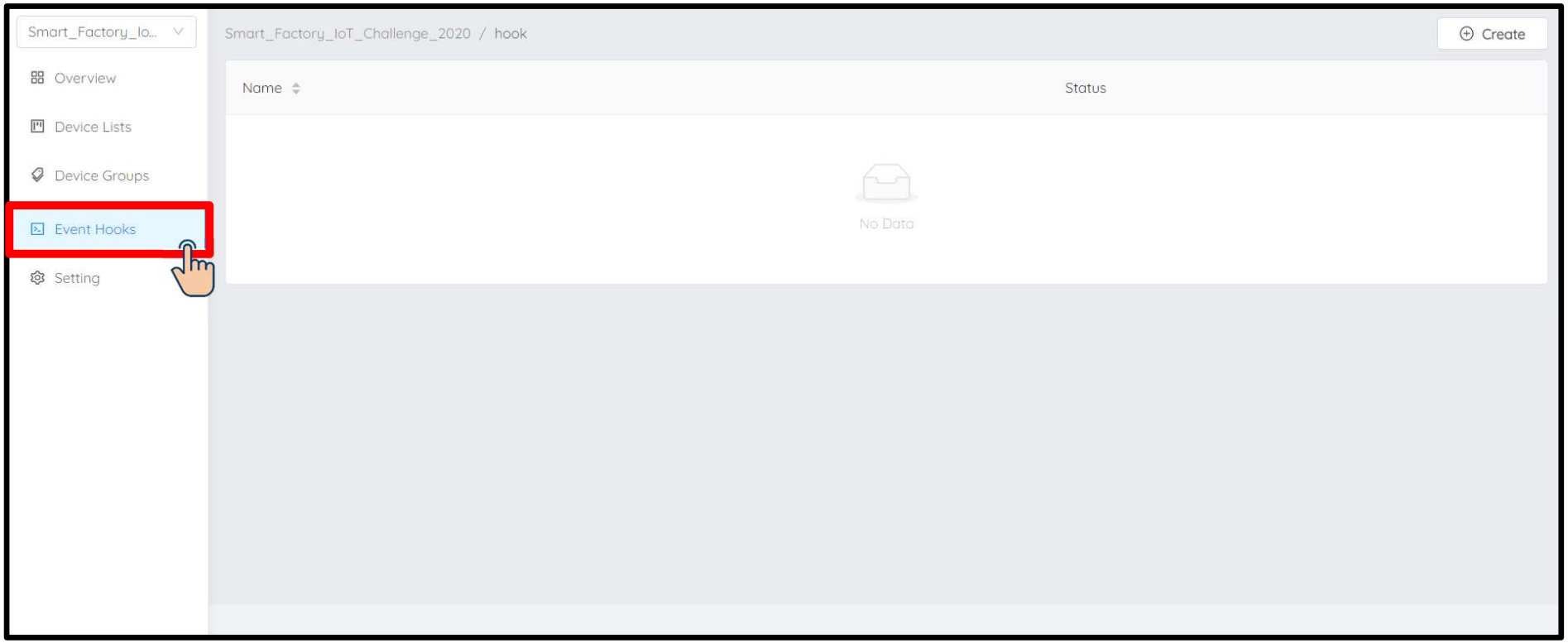
Device2 Shadow

"Temp" : 75
"Humi" : 58
Status : Offline

Trigger : DEVICE.STATUSCHANGE

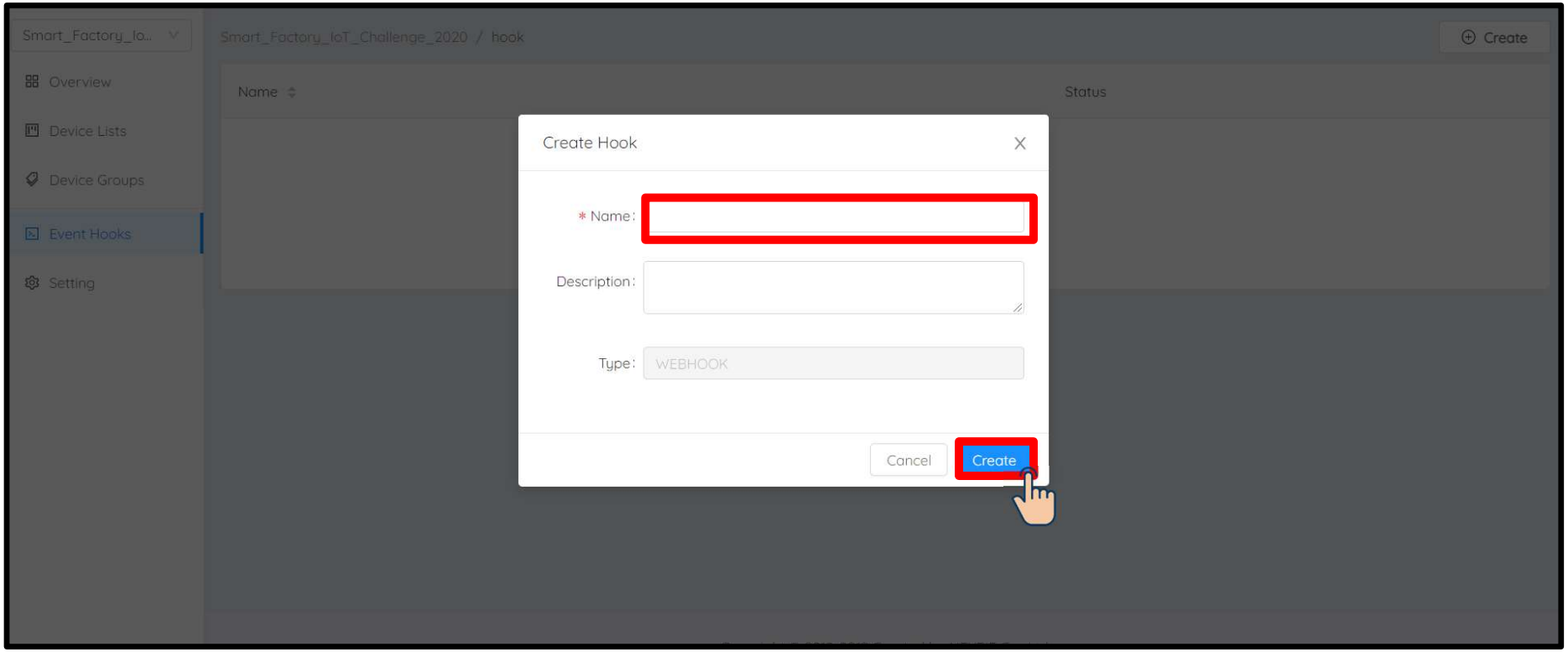
Device Data Management

Create Event Hook



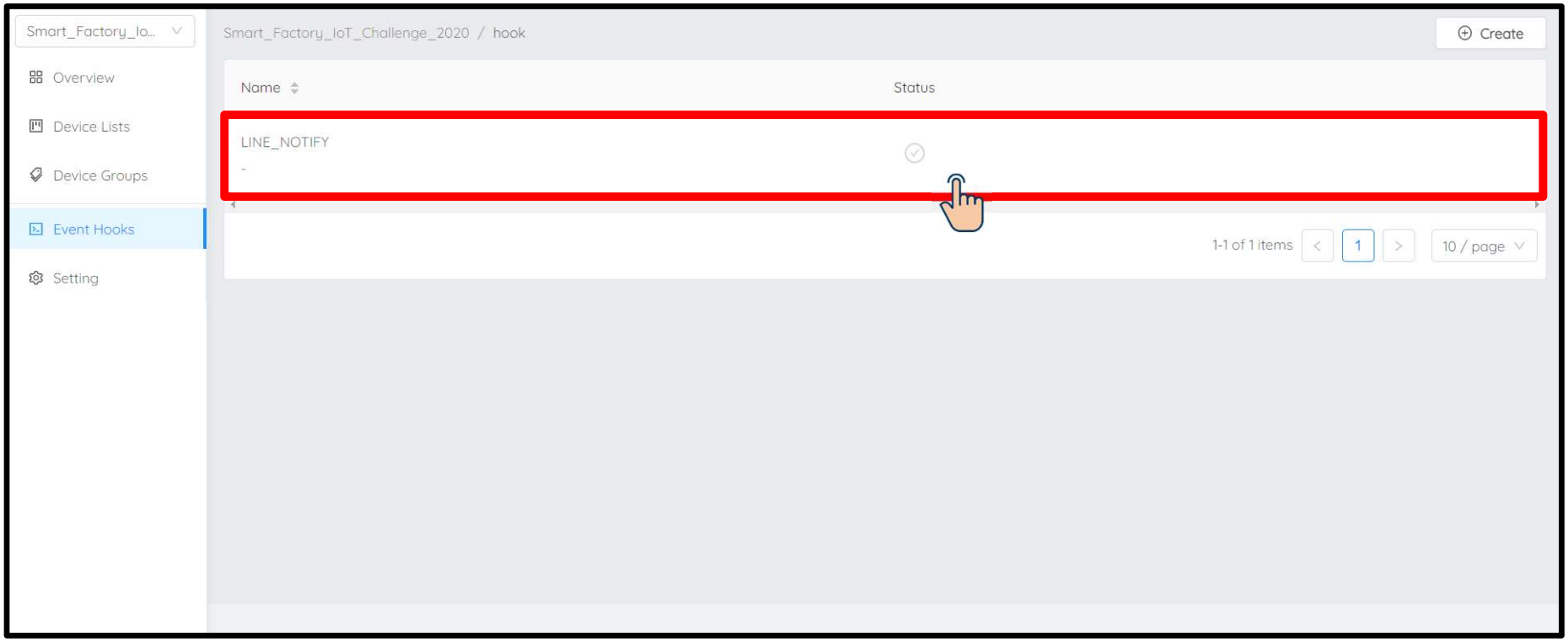
Device Data Management

Create Event Hook



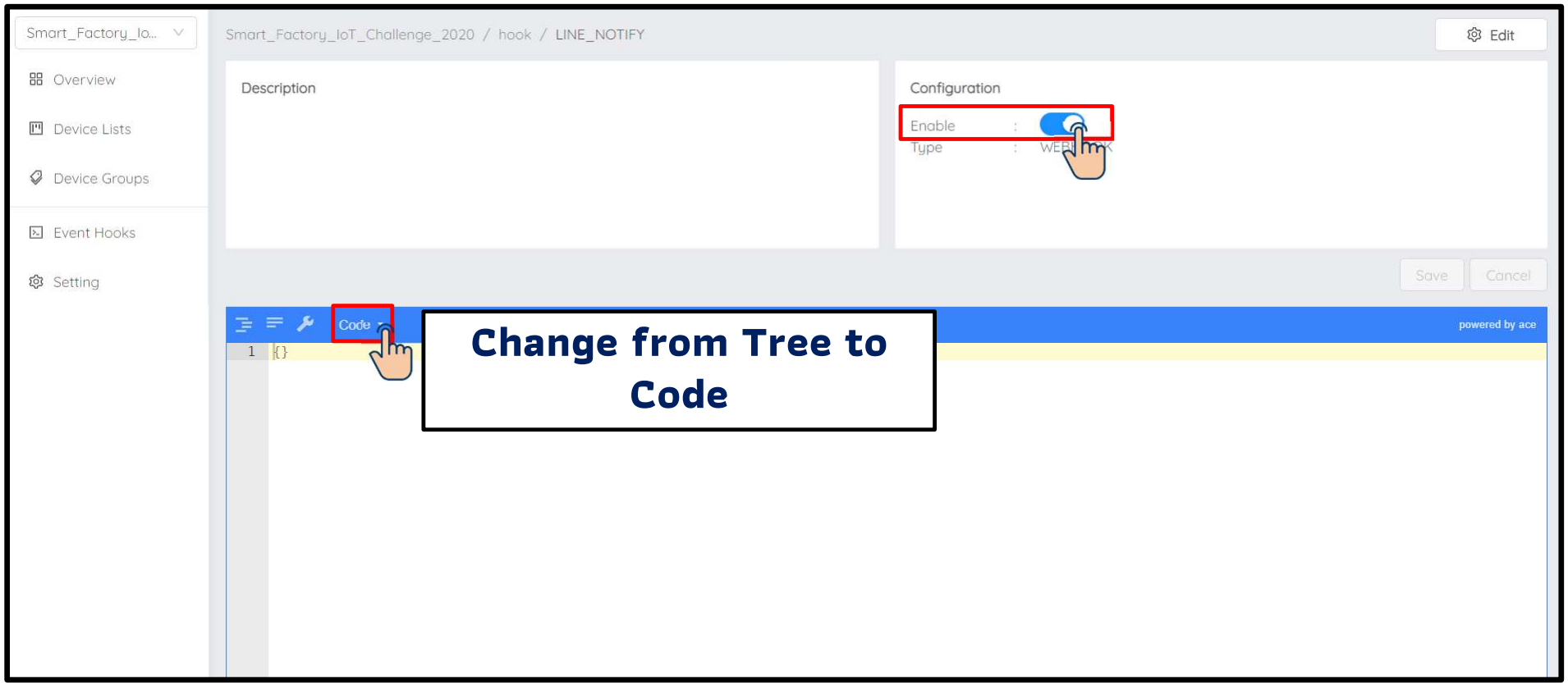
Device Data Management

Create Event Hook



Device Data Management

Create Event Hook



Create Event Hook

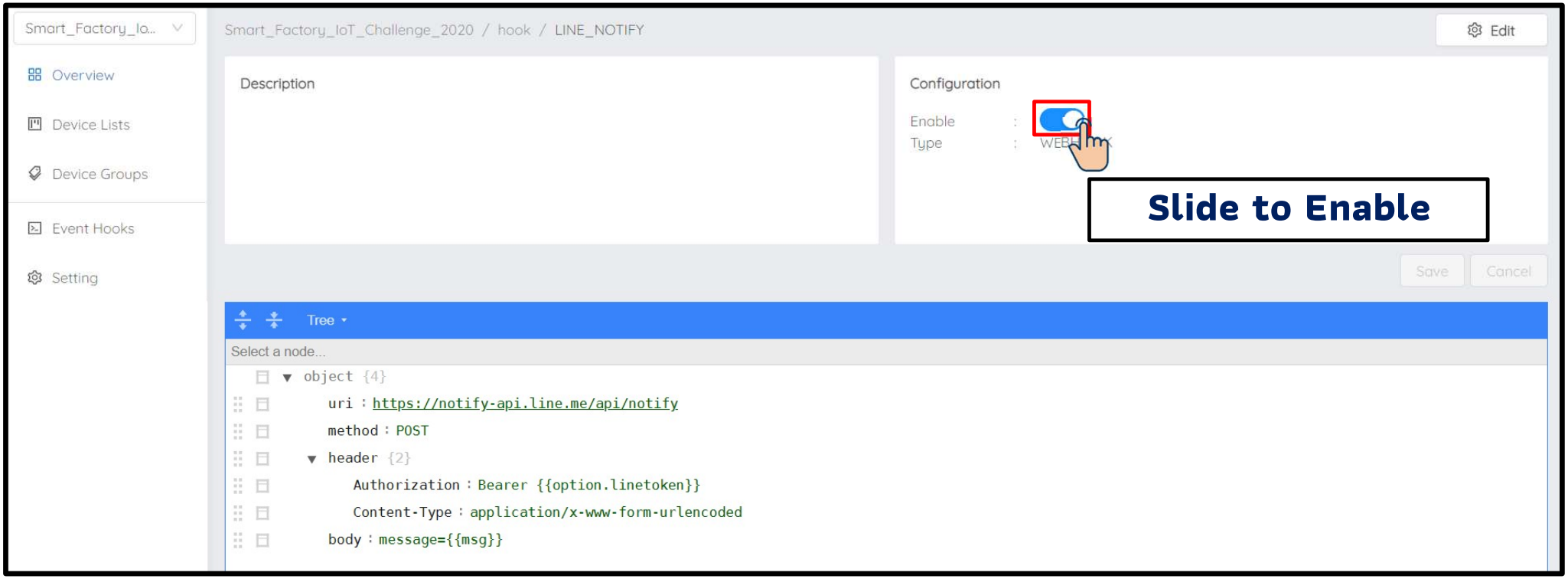
Event Hook Example of a Line Notify that specifies 4 Attributes

```
{
   "body": "message={{msg}}",
   "header": {
      "Authorization": "Bearer {{option.linetoken}}",
      "Content-Type": "application/x-www-form-urlencoded"
   },
   "method": "POST",
   "uri": "https://notify-api.line.me/api/notify"
}
```

Copy and paste to Event Hook on NETPIE2020

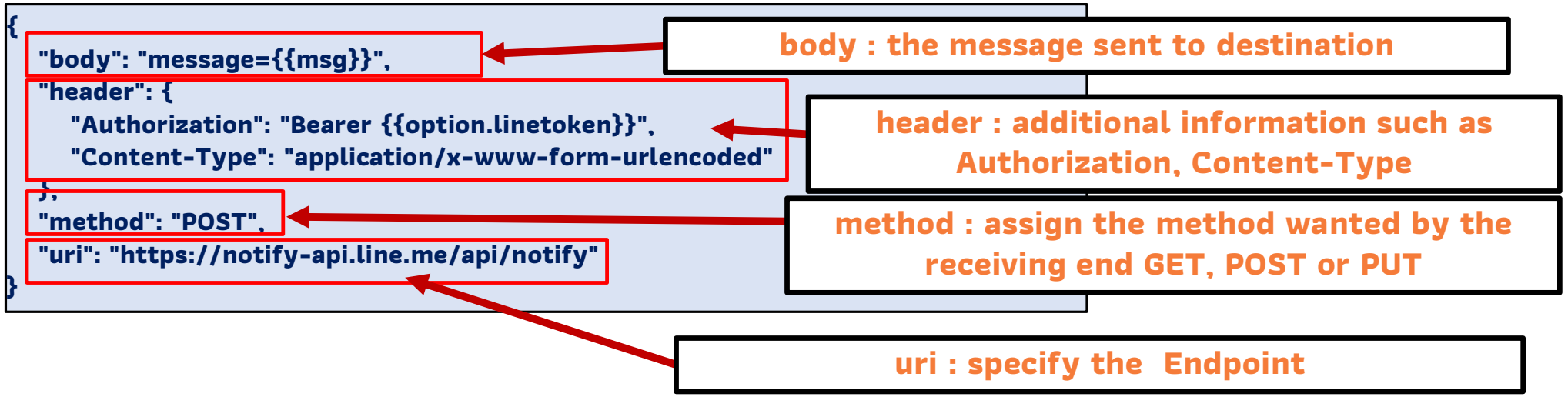
Device Data Management

Create Event Hook



Create Event Hook

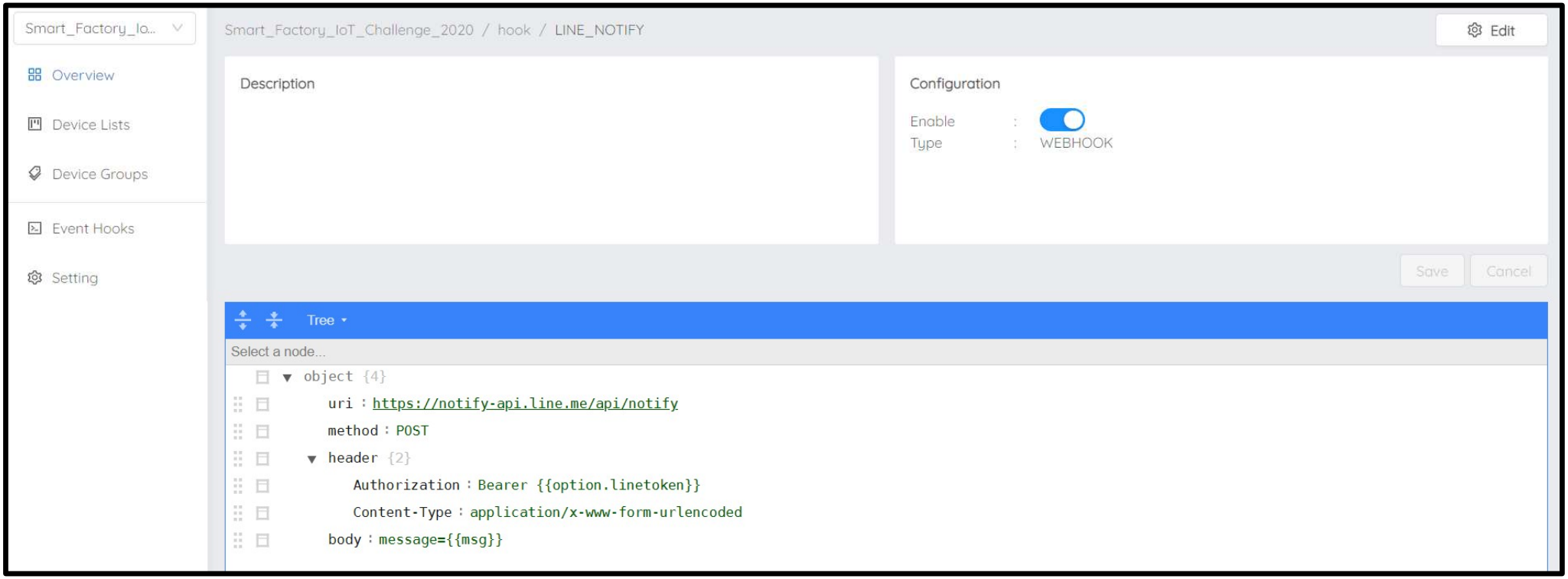
Event Hook Example of a Line Notify that specifies 4 Attributes



*** In Event Hook, variables sent from Trigger can be referenced by using {{…}} symbols around them. For example, if we want to reference msg from Trigger, use {{msg}} Or in the option of linetoken, use {{option.linetoken}}

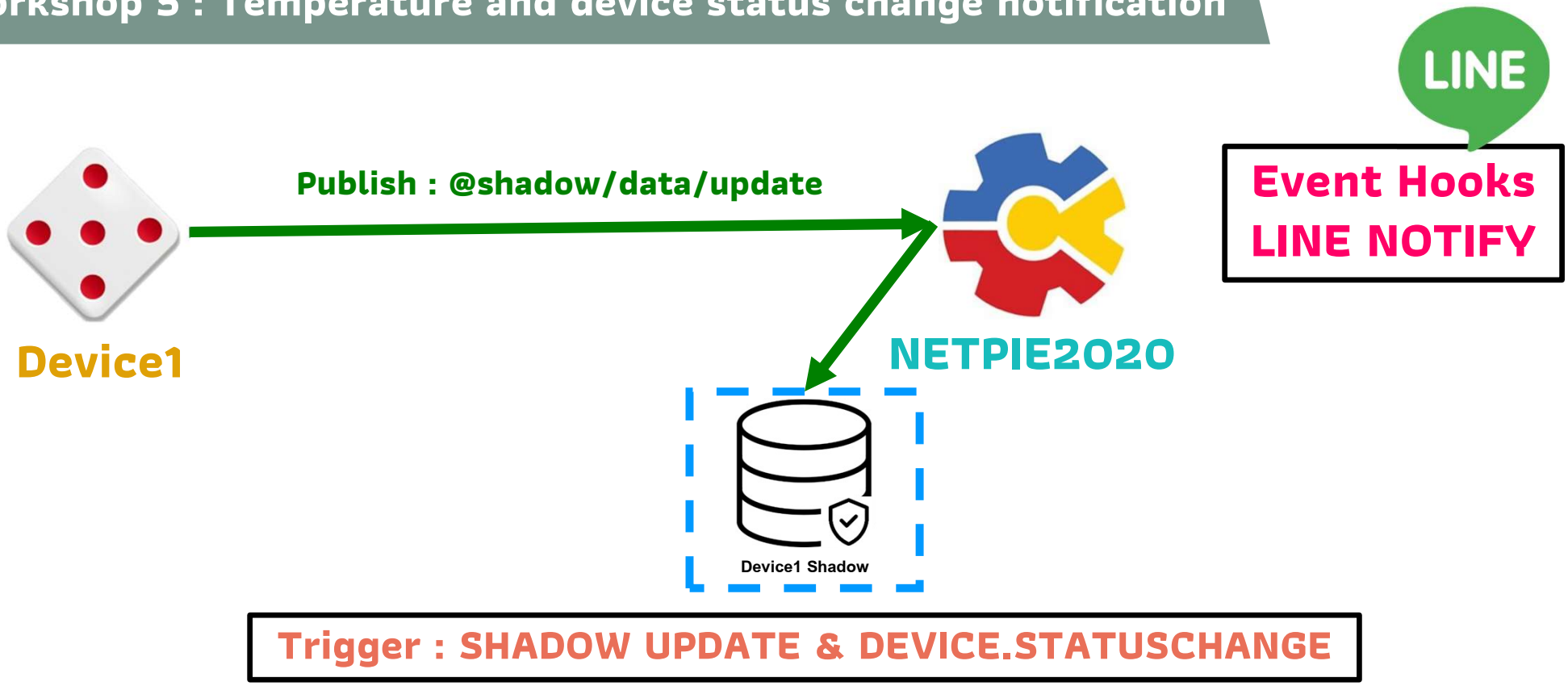
Device Data Management

Create Event Hook



Device Data Management

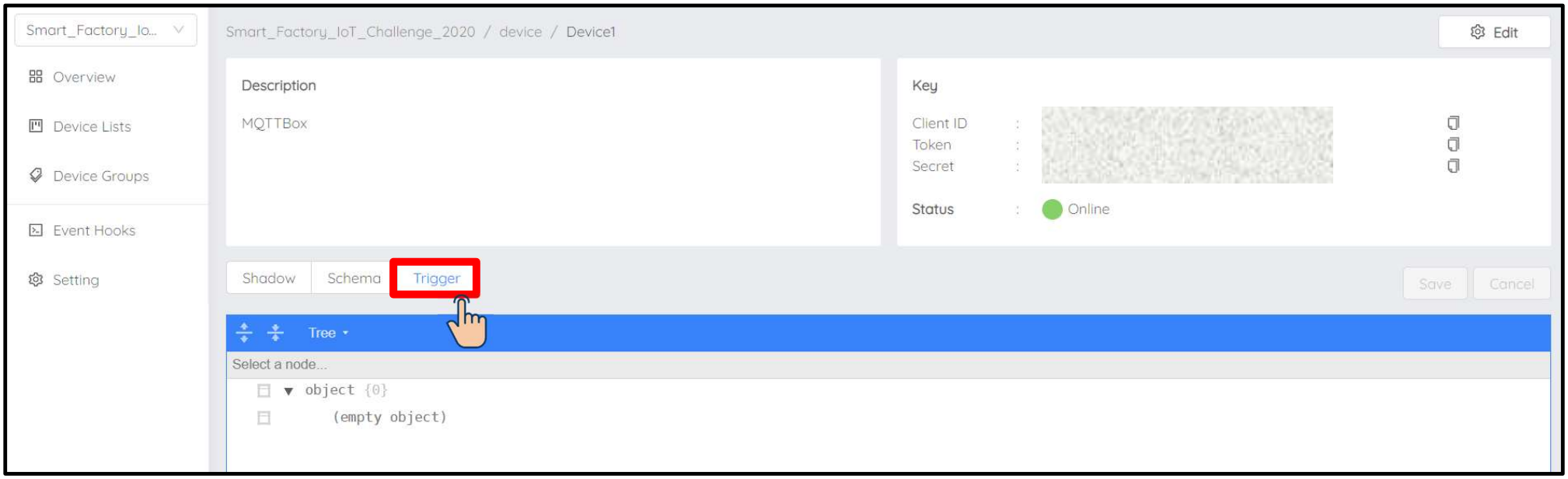
Workshop 5 : Temperature and device status change notification



Device Data Management

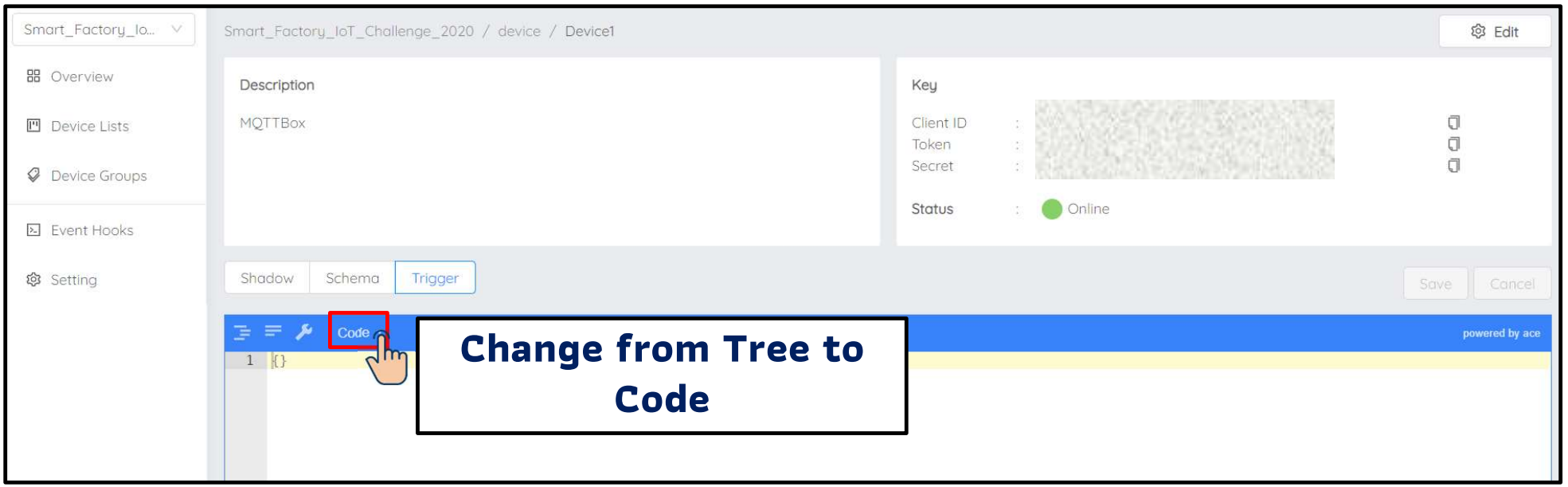
Workshop 5 : Temperature and device status change notification

Select Device1 to create Trigger



Device Data Management

Workshop 5 : Temperature and device status change notification



Device Data Management

Workshop 5 : Temperature and device status change notification

```
{
 "enabled": true,
 "trigger": [
  {
   "action": "LINE_NOTIFY",
   "event": "SHADOW.UPDATED",
   "condition": "$.temperature!= $$.temperature",
   "msg": "My temperature 2 was change from {{$$.temperature}} to {{$.temperature}}",
   "option": {
    "linetoken": "Your_Token_LINE"
   }
  },
  {
   "action": "LINE_NOTIFY",
   "event": "DEVICE.STATUSCHANGED",
   "msg": "My Device {{$.statustext}}, statuscode: {{$.status}}",
   "option": {
    "linetoken": "Your_Token_LINE"
   }
  }
 ]
}
```

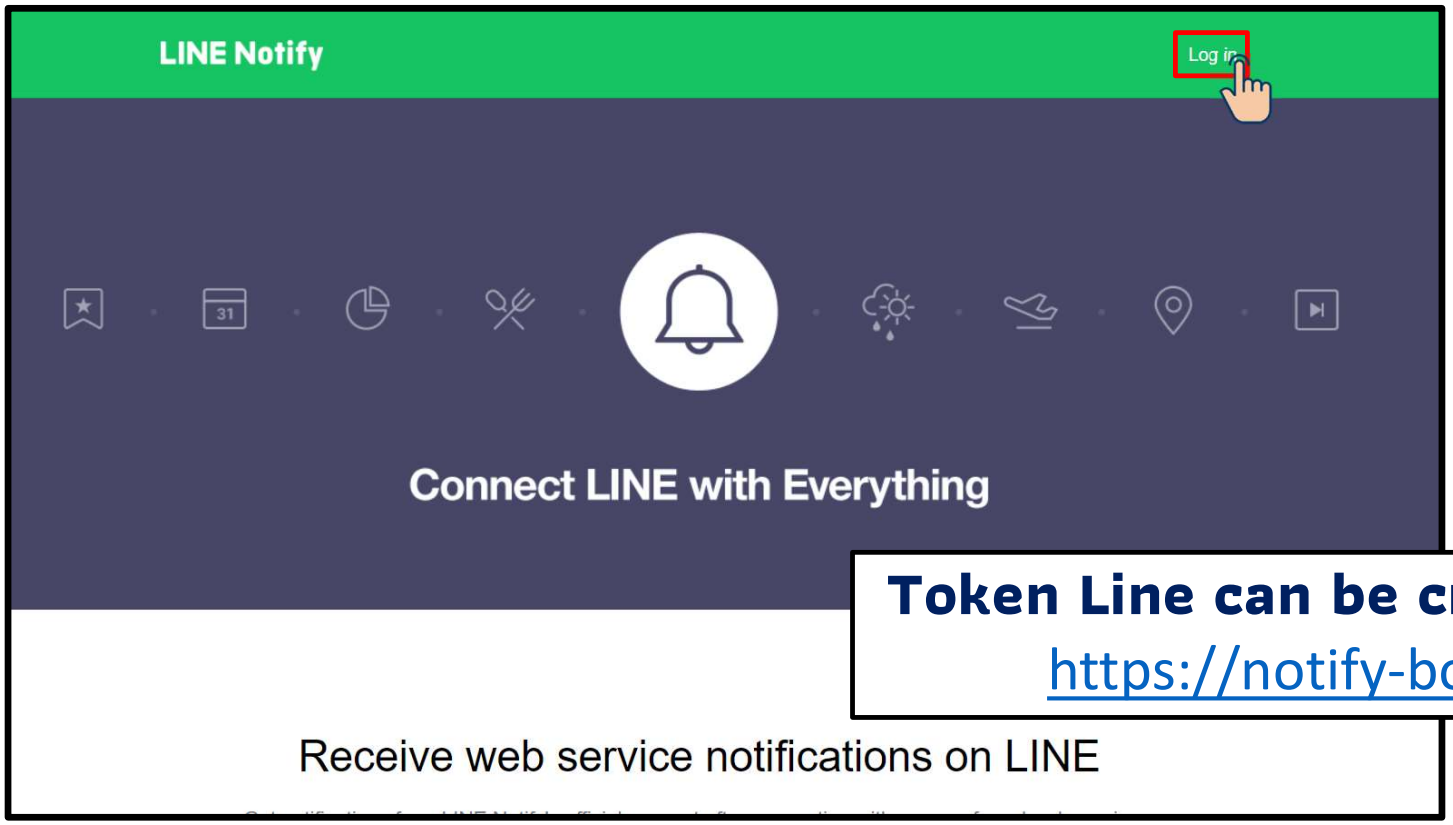
Turn on Trigger

Copy this trigger code and paste to Device Trigger

Trigger Shadow Update : the condition is to take action whenever the temperature changes, with action : “LINE_NOTIFY” and the message is “My temperature 2 was changed from xx to xx.”

Trigger Status Change : The condition is to take action whenever device status changes, with action: “LINE_NOTIFY” and the message is “My device xx statuccode: xx”

Workshop 5 : Temperature and device status change notification

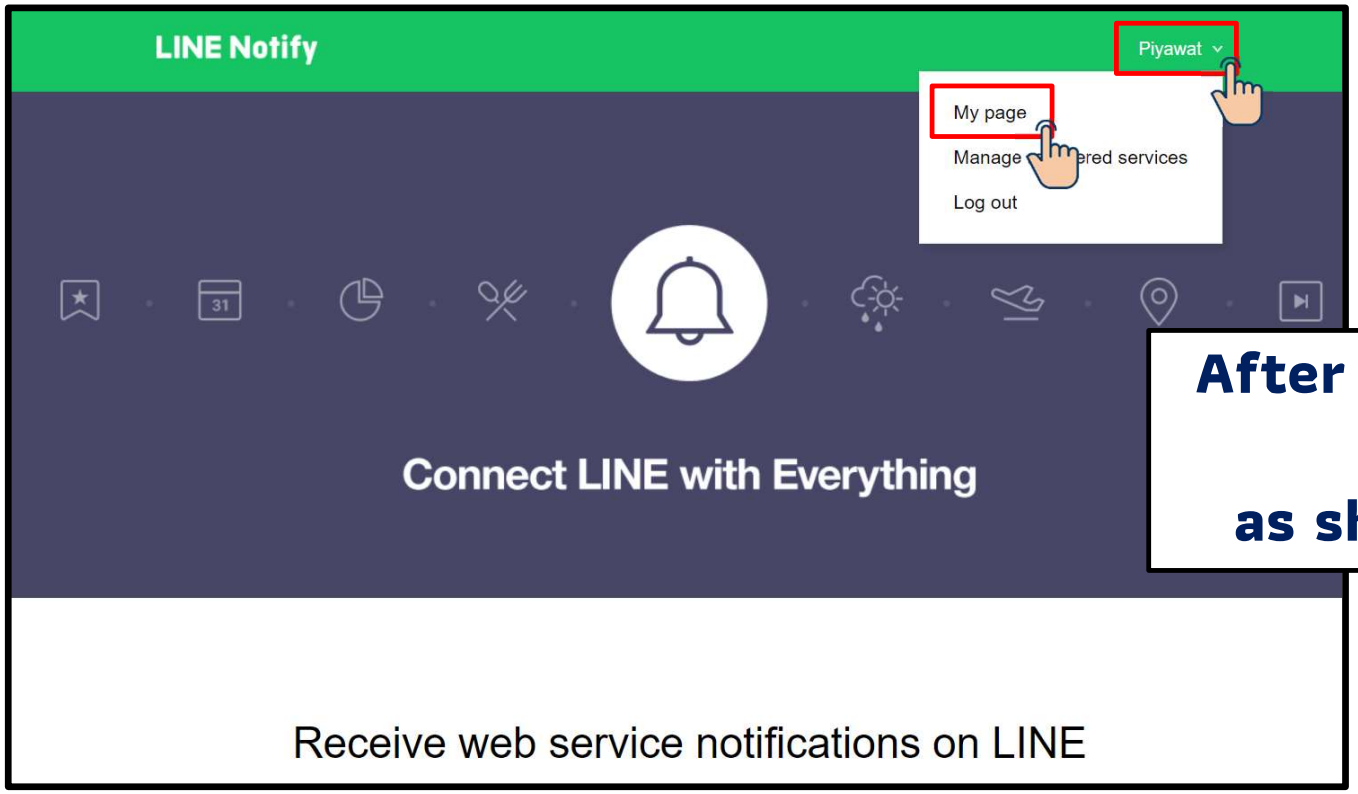


Token Line can be created at this link
https://notify-bot.line.me/en/

Workshop 5 : Temperature and device status change notification

Login to your LINE account using E-mail and Password

Workshop 5 : Temperature and device status change notification

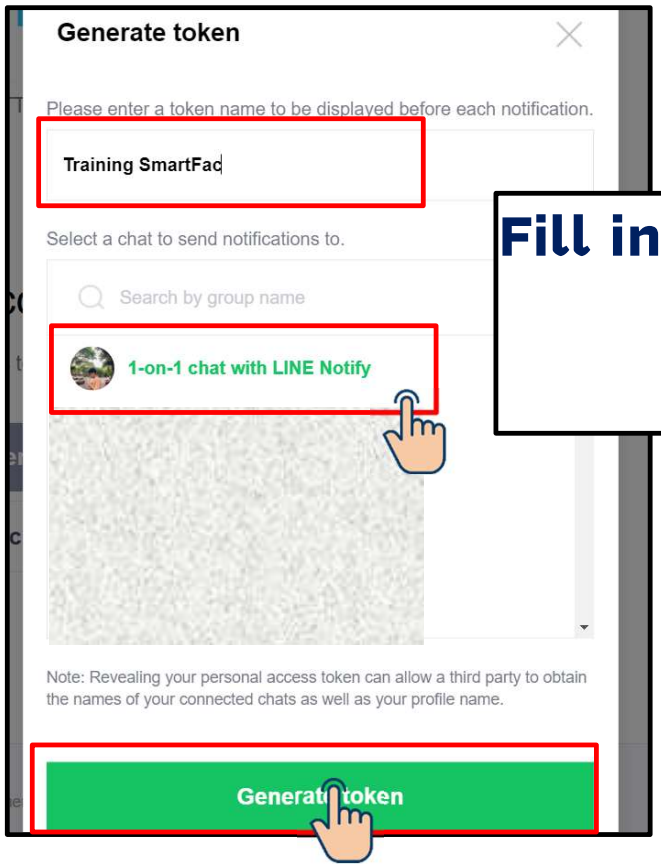


After login, select >> My page as shown in the figure

Workshop 5 : Temperature and device status change notification

Scroll down and click on [Generate Token]

Workshop 5 : Temperature and device status change notification



Fill in token name, and select 1-on-1 chat with LINE notify.
Click [Generate Token]

Workshop 5 : Temperature and device status change notification

The generated token is displayed. Copy and paste to linetoken in Device Trigger

Workshop 5 : Temperature and device status change notification

Shadow | Schema | Trigger

Save | Cancel

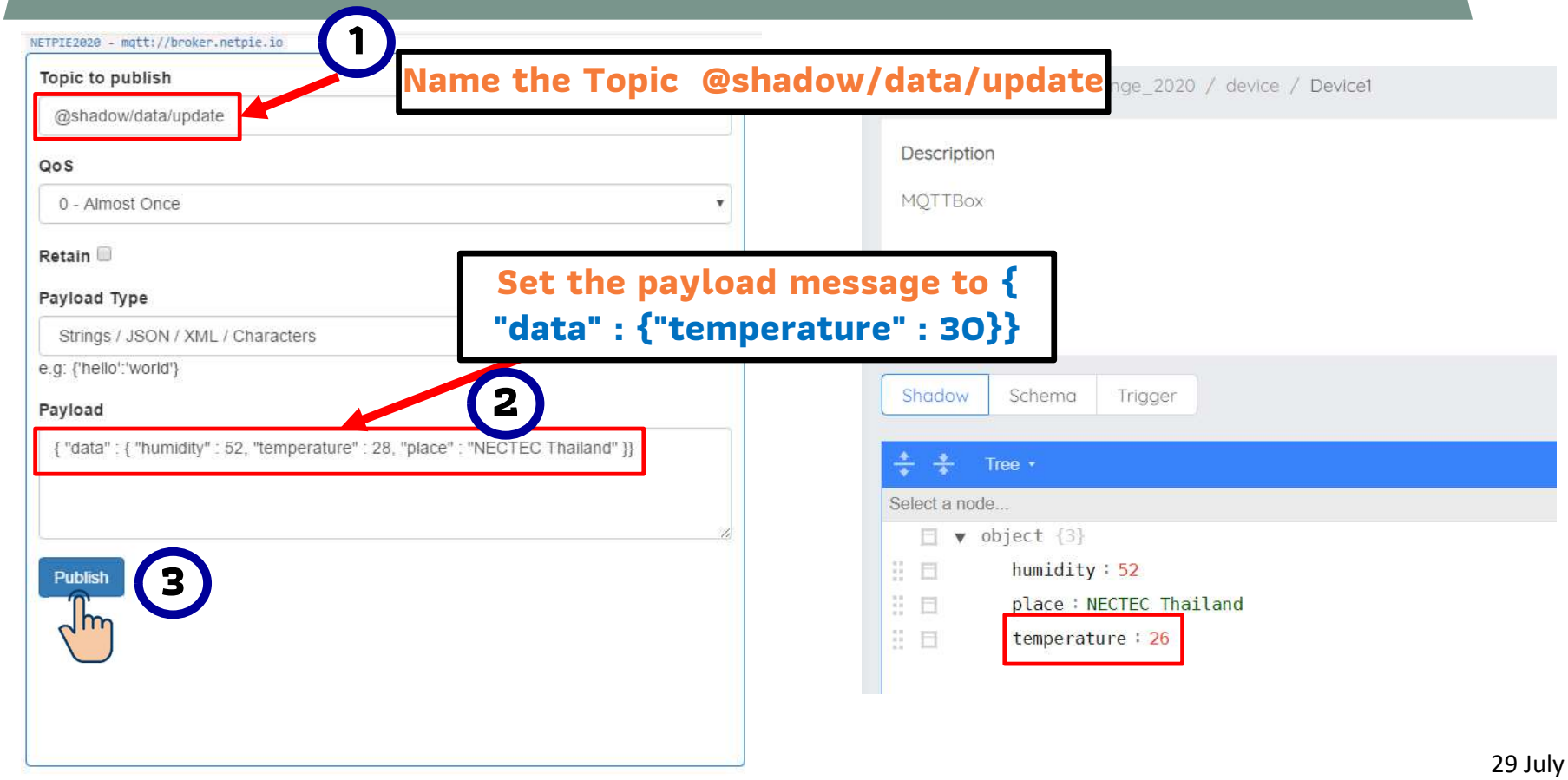
Code

```
{
  "enabled": true,
  "trigger": [
    {
      "action": "LINE_NOTIFY",
      "event": "SHADOW.UPDATED",
      "condition": "$.temperature!= $$.temperature",
      "msg": "My temperature 2 was change from {{$$.temperature}} to {{$.temperature}}",
      "option": {
        "linetoken": "Your_Token_LINE"
      }
    },
    {
      "action": "LINE_NOTIFY",
      "event": "DEVICE.STATUSCHANGED",
      "msg": "My Device {{$.statustext}}, statuscode: {{$.status}}",
      "option": {
        "linetoken": "Your_Token_LINE"
      }
    }
  ]
}
```

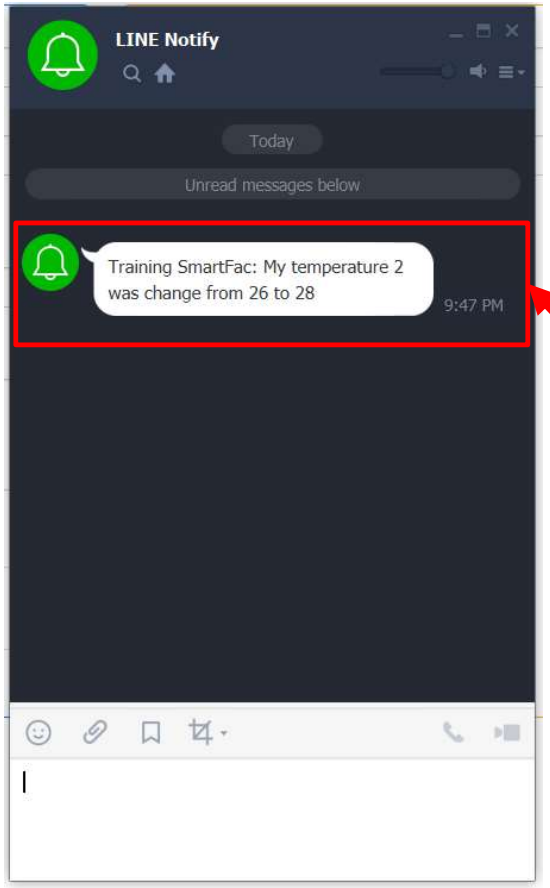
Ln:22 Col:2

Device Data Management

Workshop 5 : Temperature and device status change notification

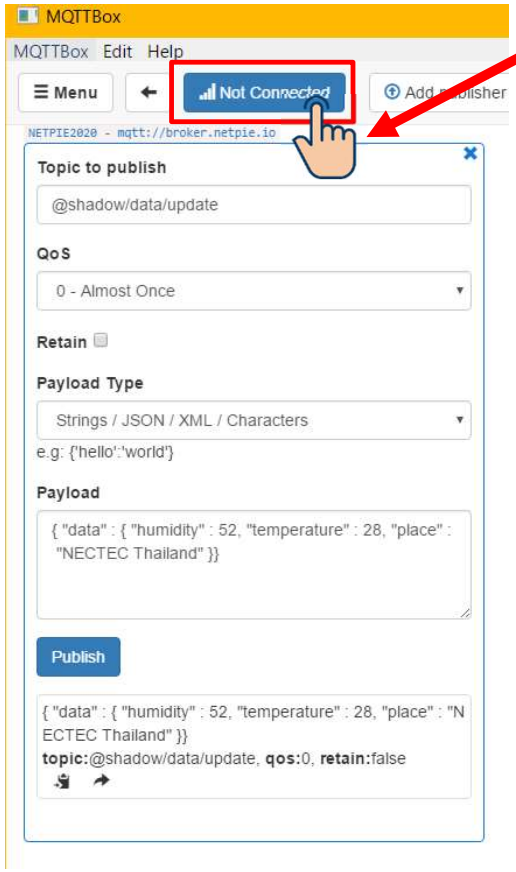


Workshop 5 : Temperature and device status change notification

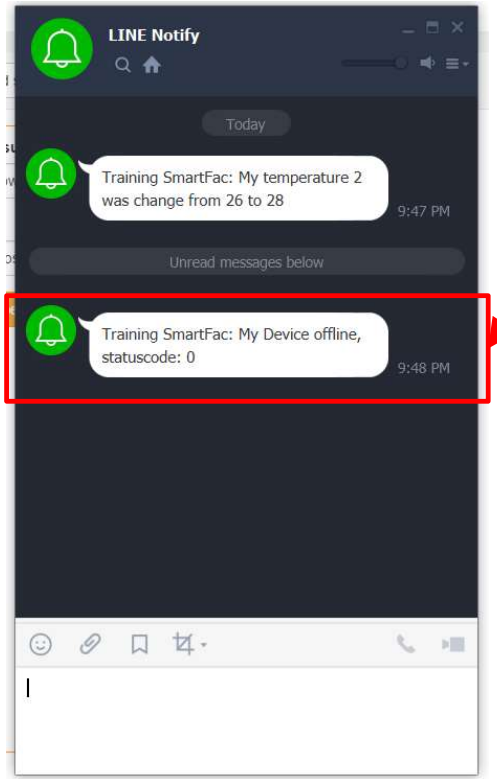


Temperature change notification from LINE

Workshop 5 : Temperature and device status change notification



Device status change test



Device status change notification LINE

NETPIE Dashboard

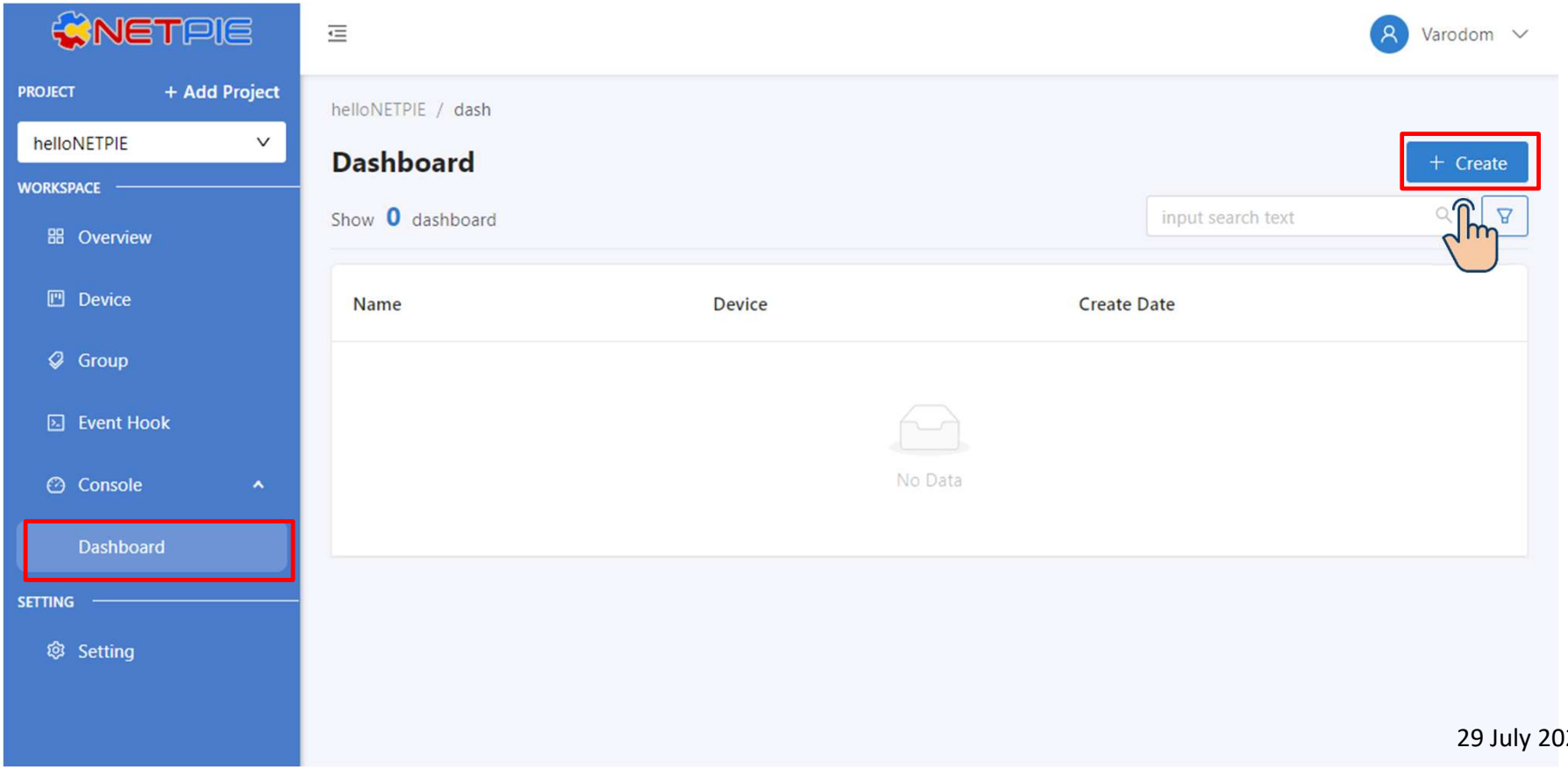
Dashboard

What is Dashboard?

NETPIE2020 Dashboard is a software panel for control and display data from a device. A developer creates and customizes Widget Plugins to suit user requirements, such as gauges, sliders, control buttons, and put Javascript commands for various actions.

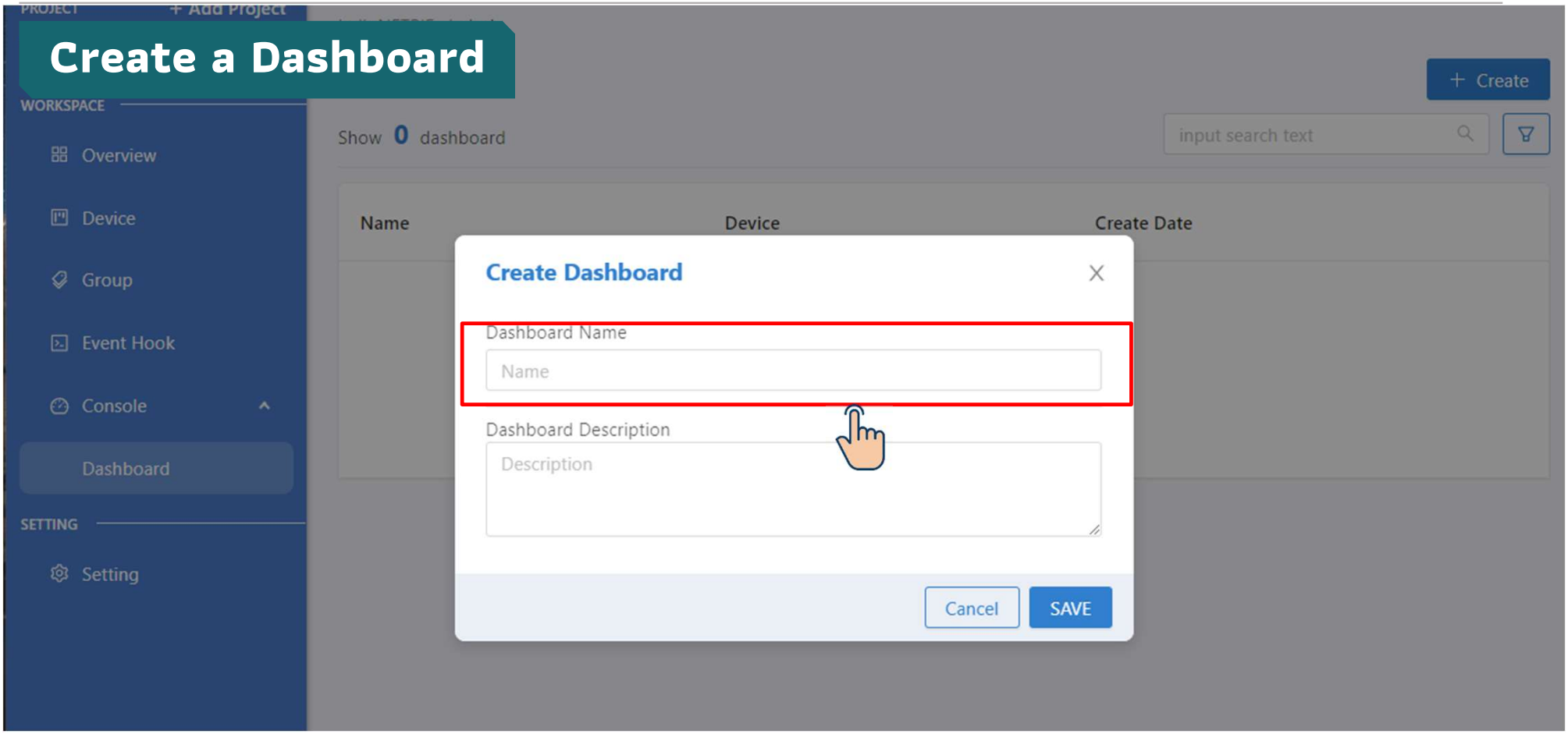
Dashboard

Create a Dashboard



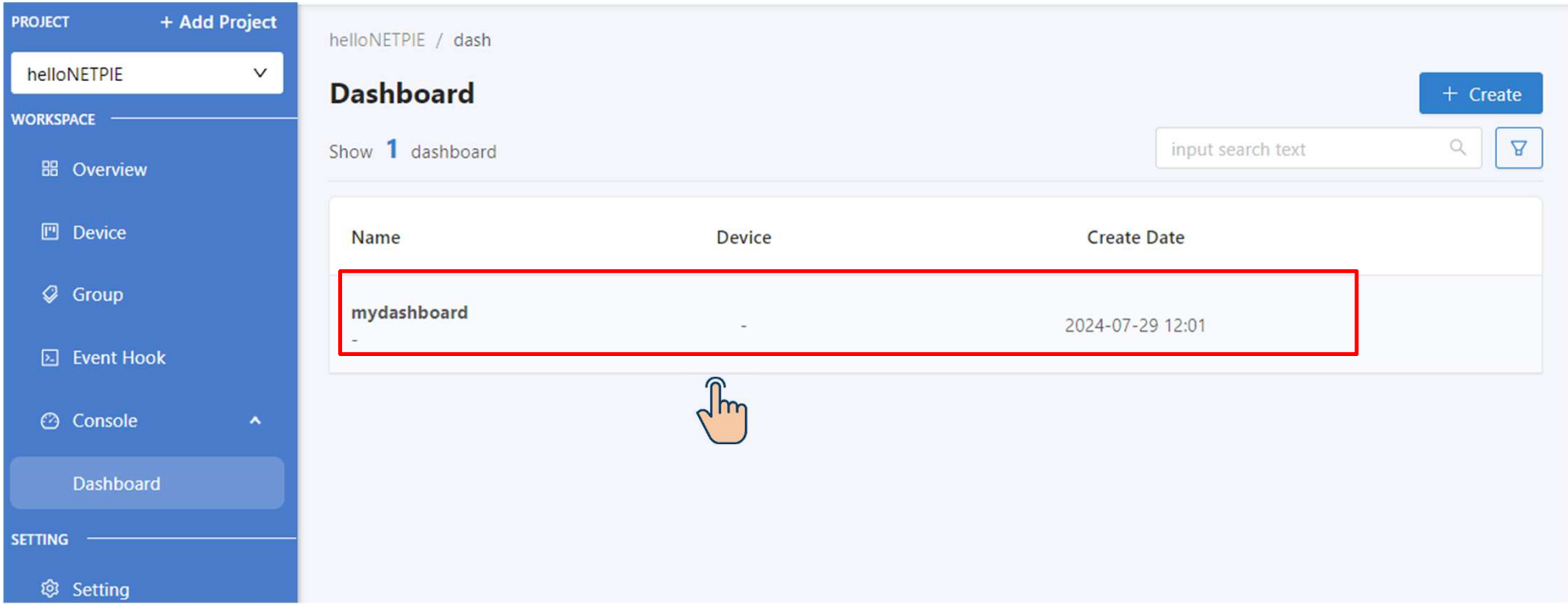
Dashboard

Create a Dashboard



Dashboard

Create a Dashboard



Dashboard

Workshop 6 : Construct a Dashboard on NETPIE2020

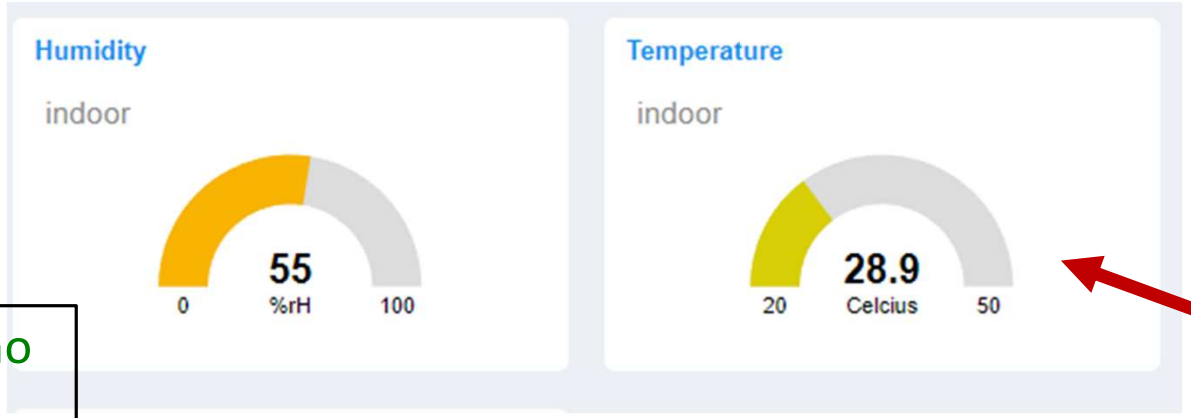
DHT11

NodeMCU

@shadow/data/update
Publish : { “data”: { “Temp” : 24, “Humi” : 58 }}

NETPIE2020

“Temp” : 75
“Humi” : 58



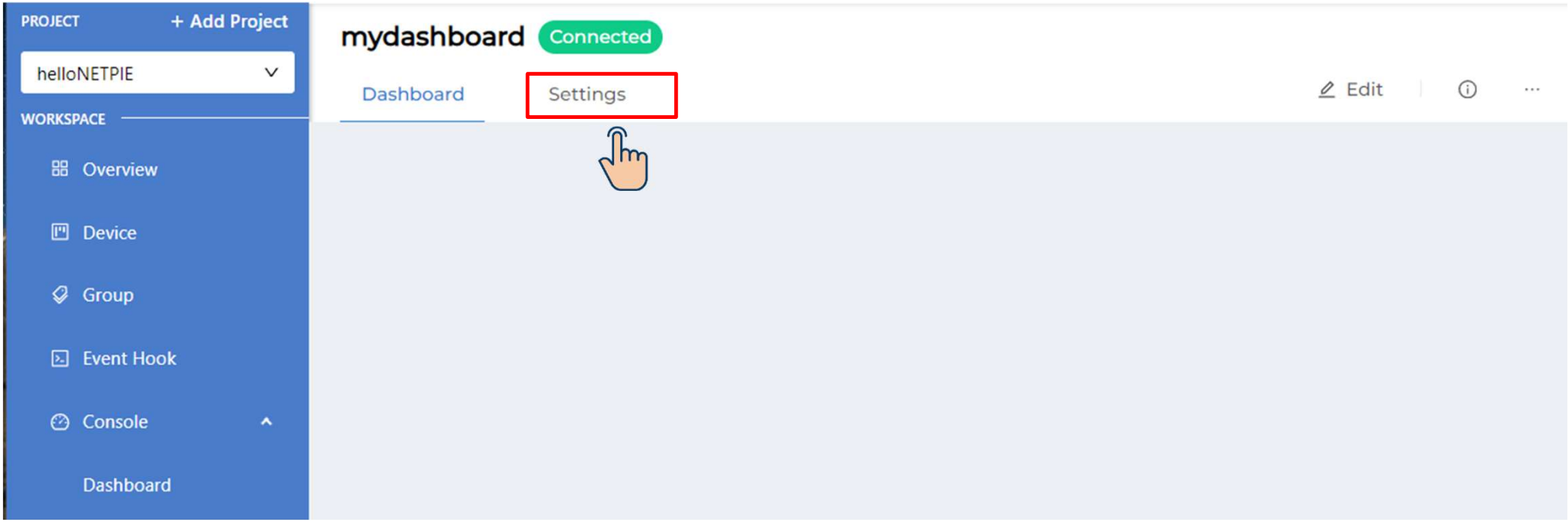
Use the Arduino Program from Workshop 4

Device Shadow

Dashboard

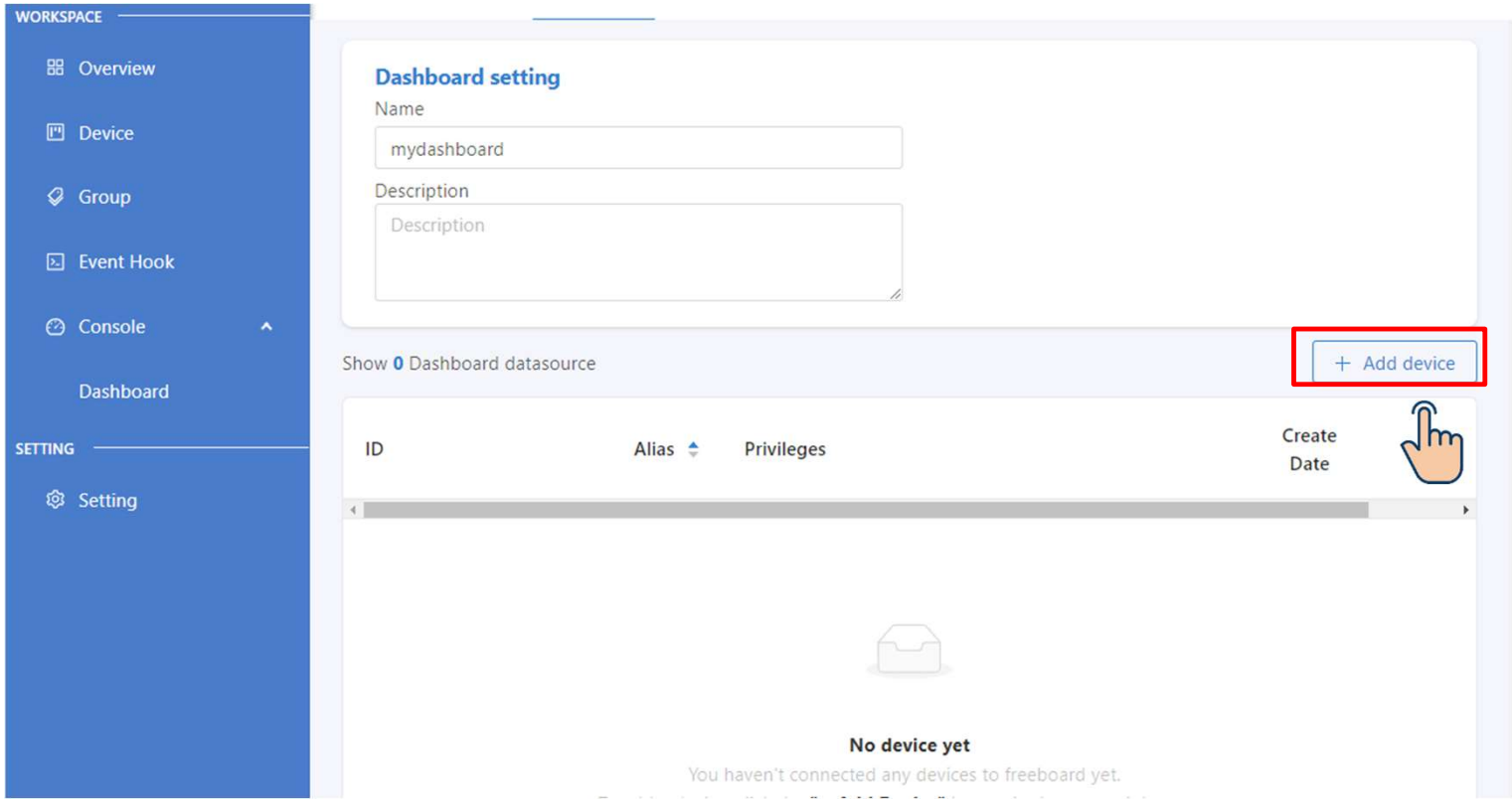
Dashboard

Workshop 6 : Construct a Dashboard on NETPIE2020



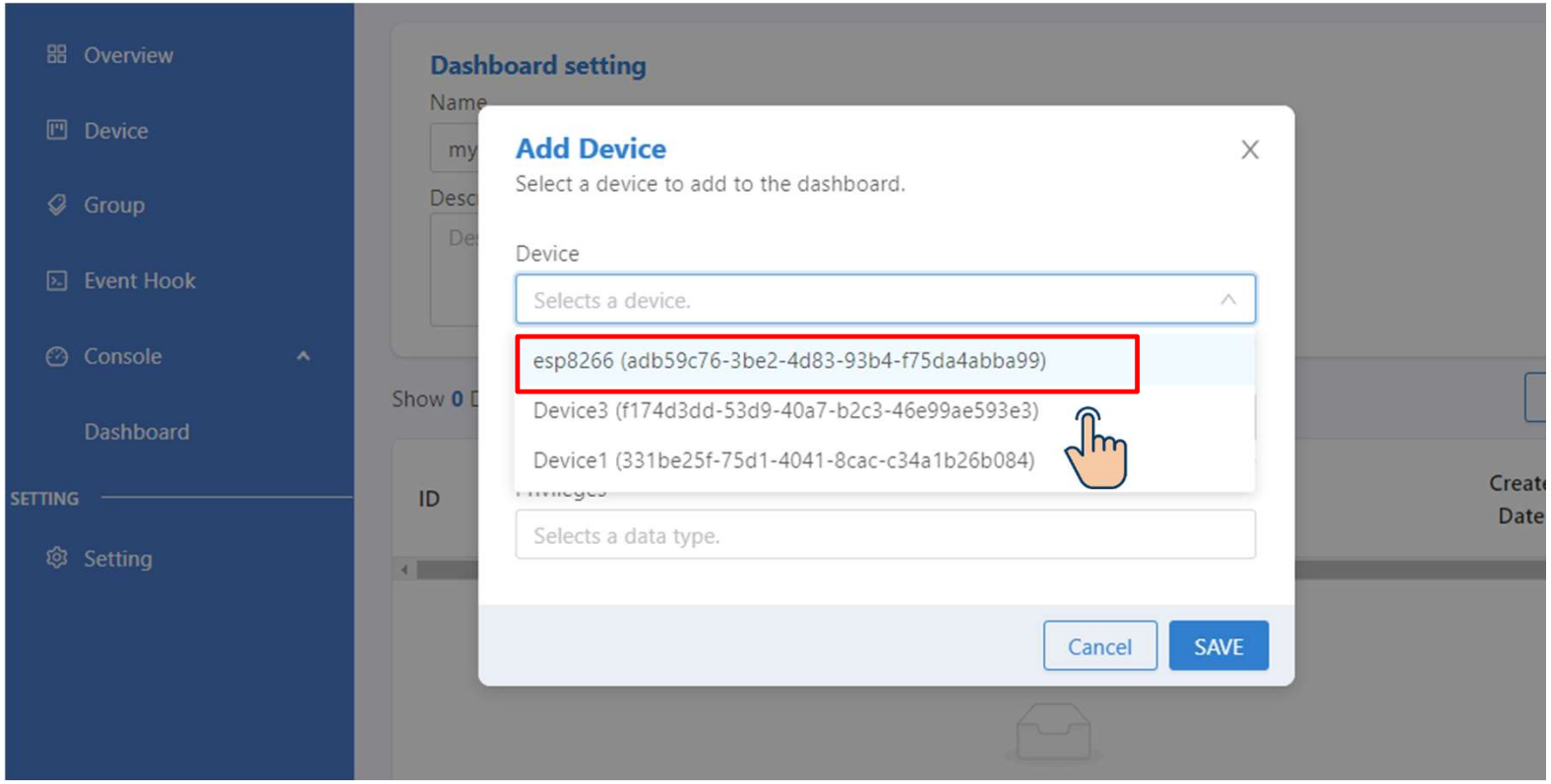
Dashboard

Workshop 6 : Construct a Dashboard on NETPIE2020



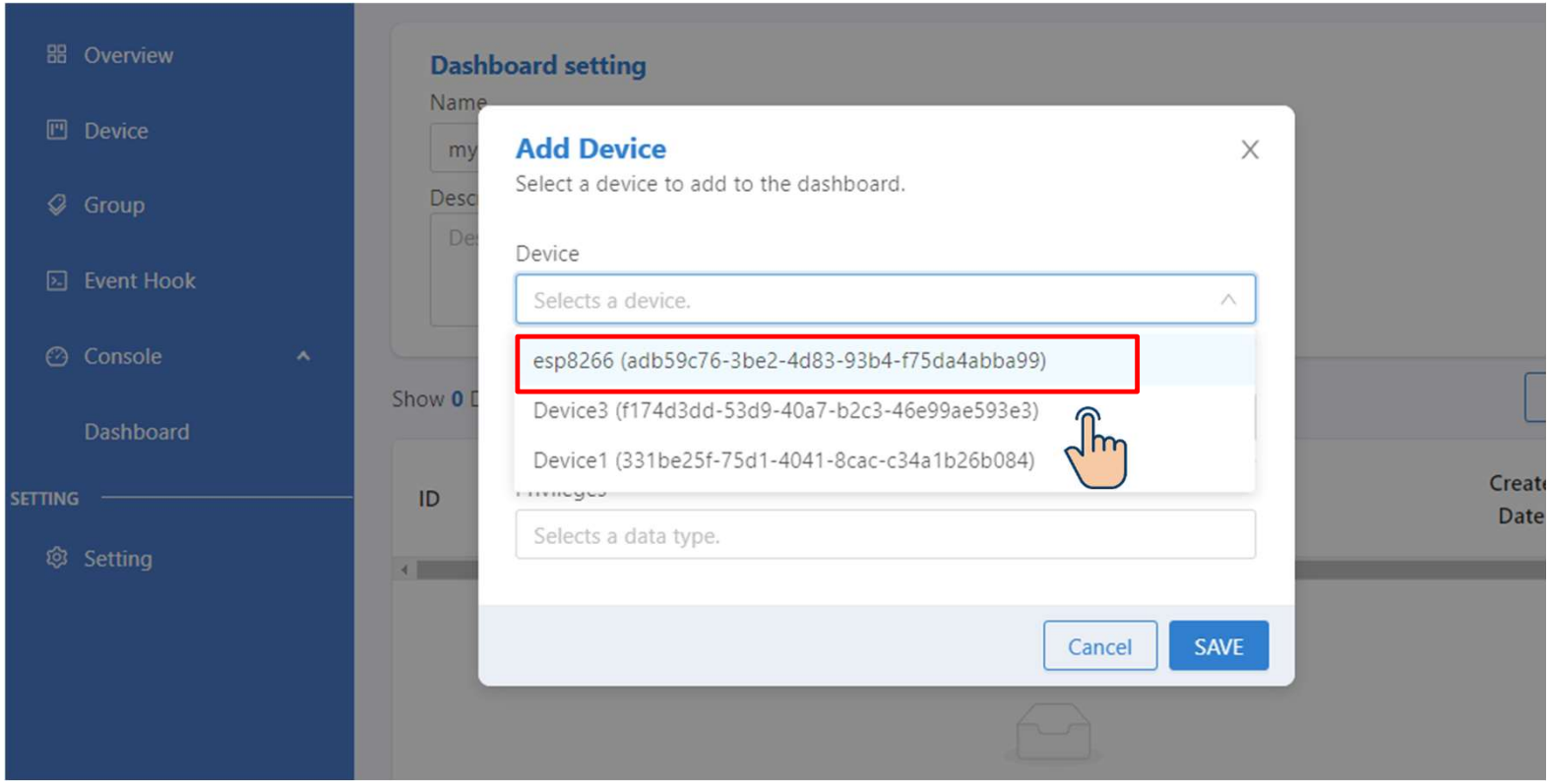
Dashboard

Workshop 6 : Construct a Dashboard on NETPIE2020



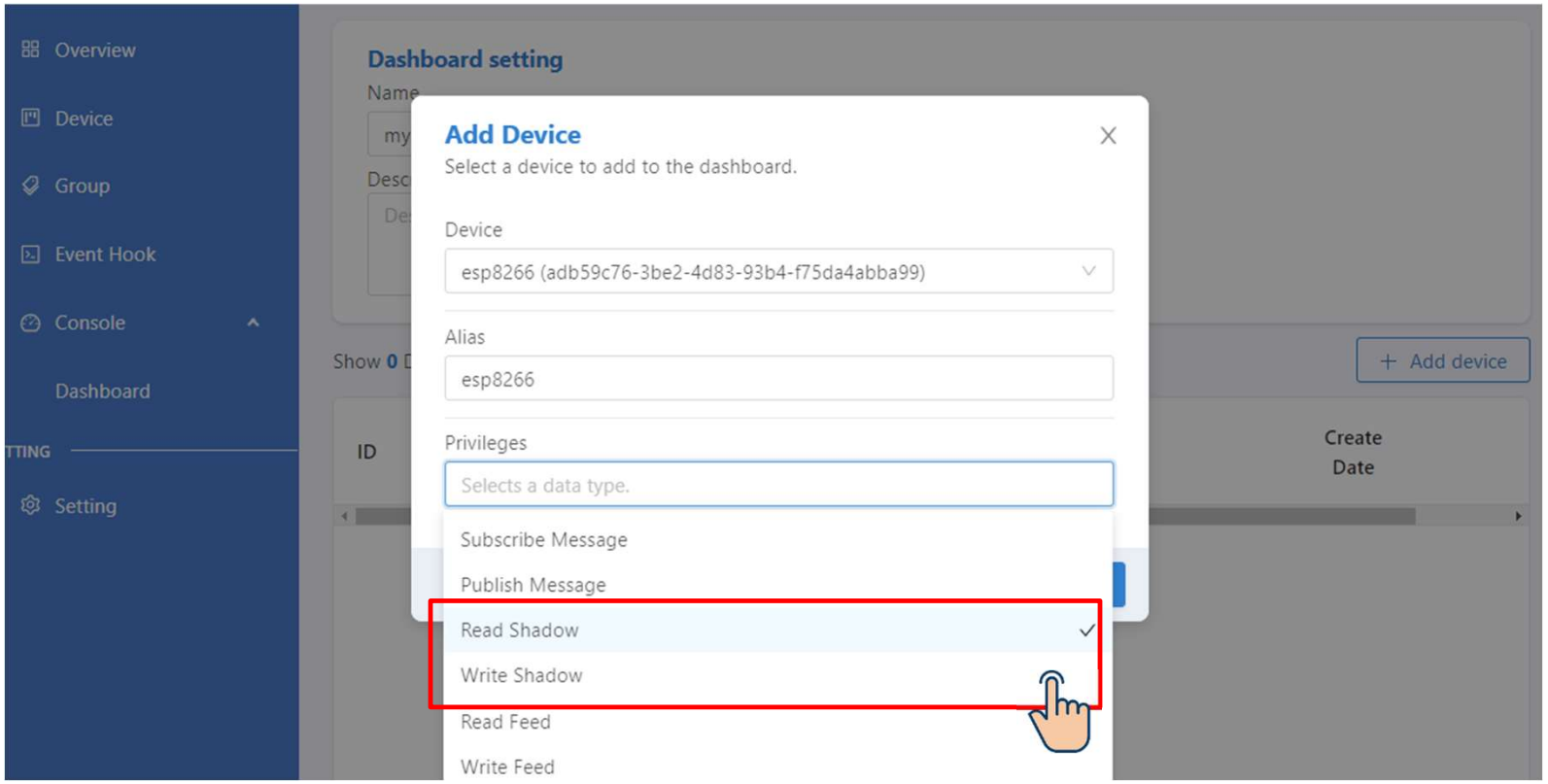
Dashboard

Workshop 6 : Construct a Dashboard on NETPIE2020



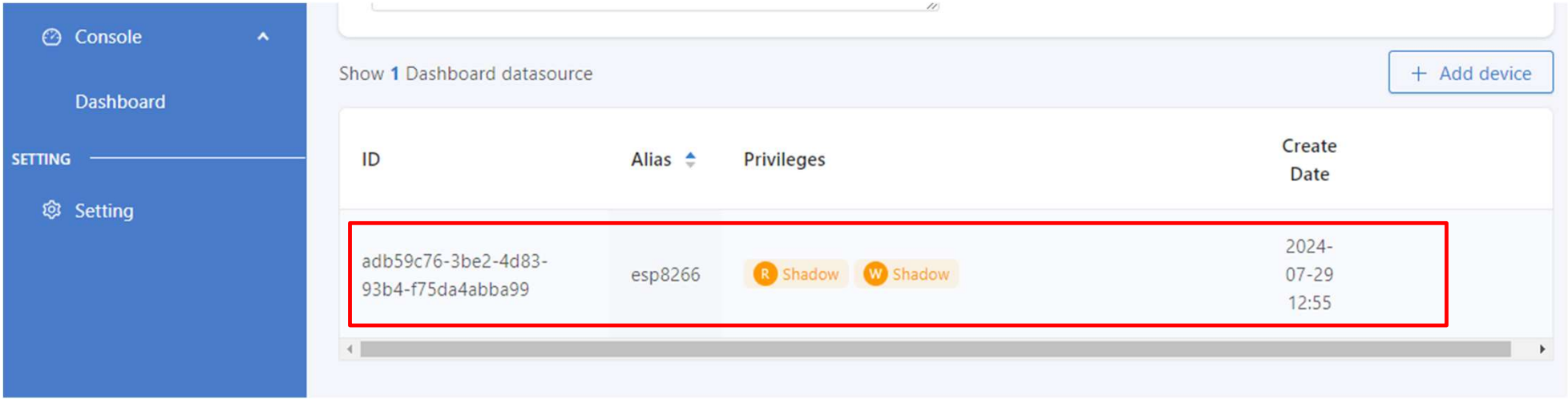
Dashboard

Workshop 6 : Construct a Dashboard on NETPIE2020



Dashboard

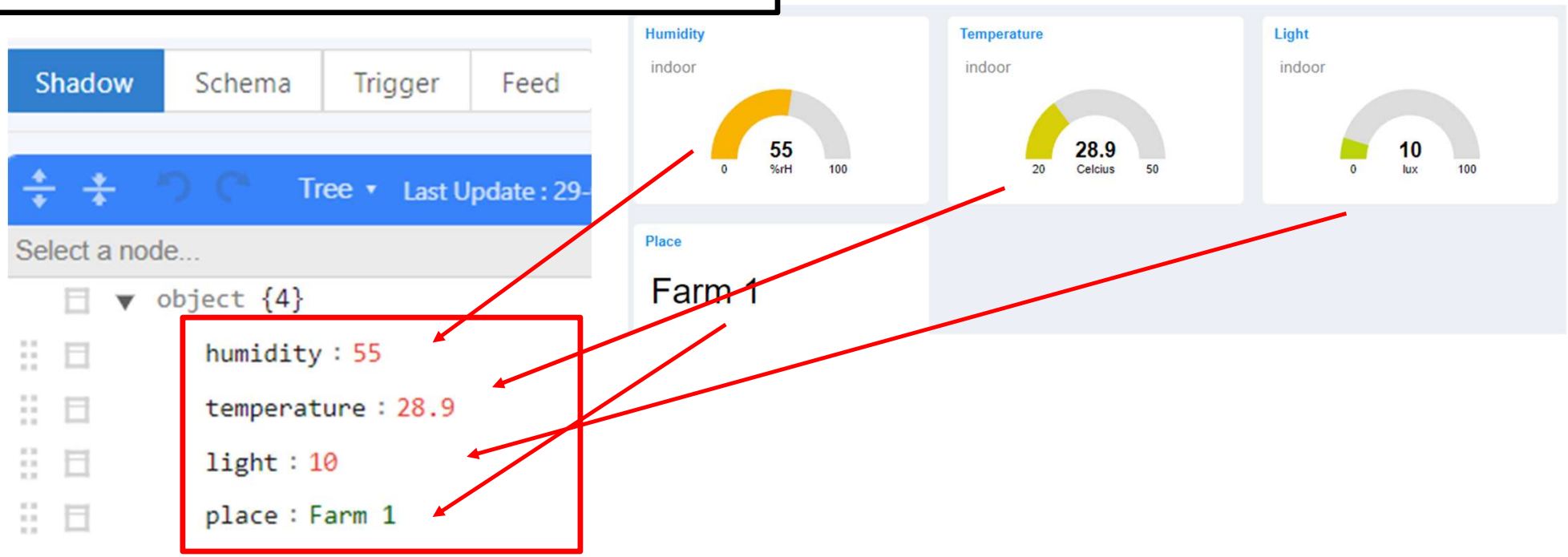
Workshop 6 : Construct a Dashboard on NETPIE2020



Dashboard

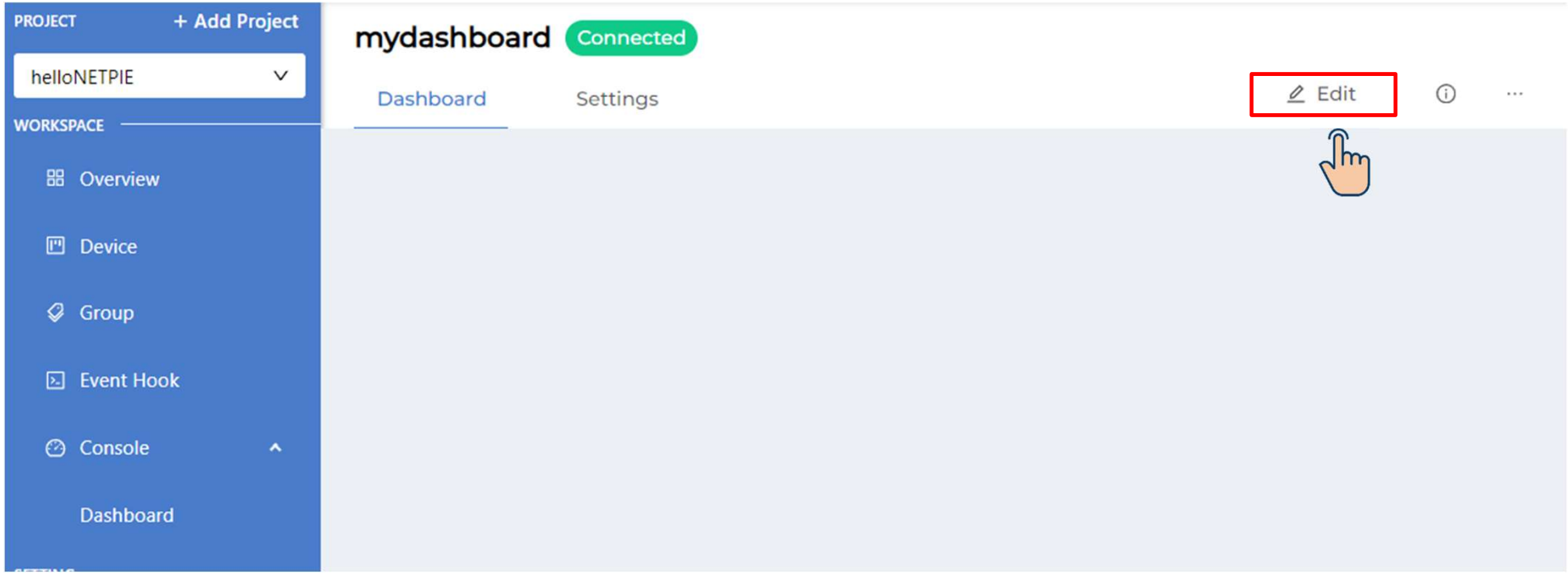
Workshop 6 : Construct a Dashboard on NETPIE2020

Data recorded in Device Shadow



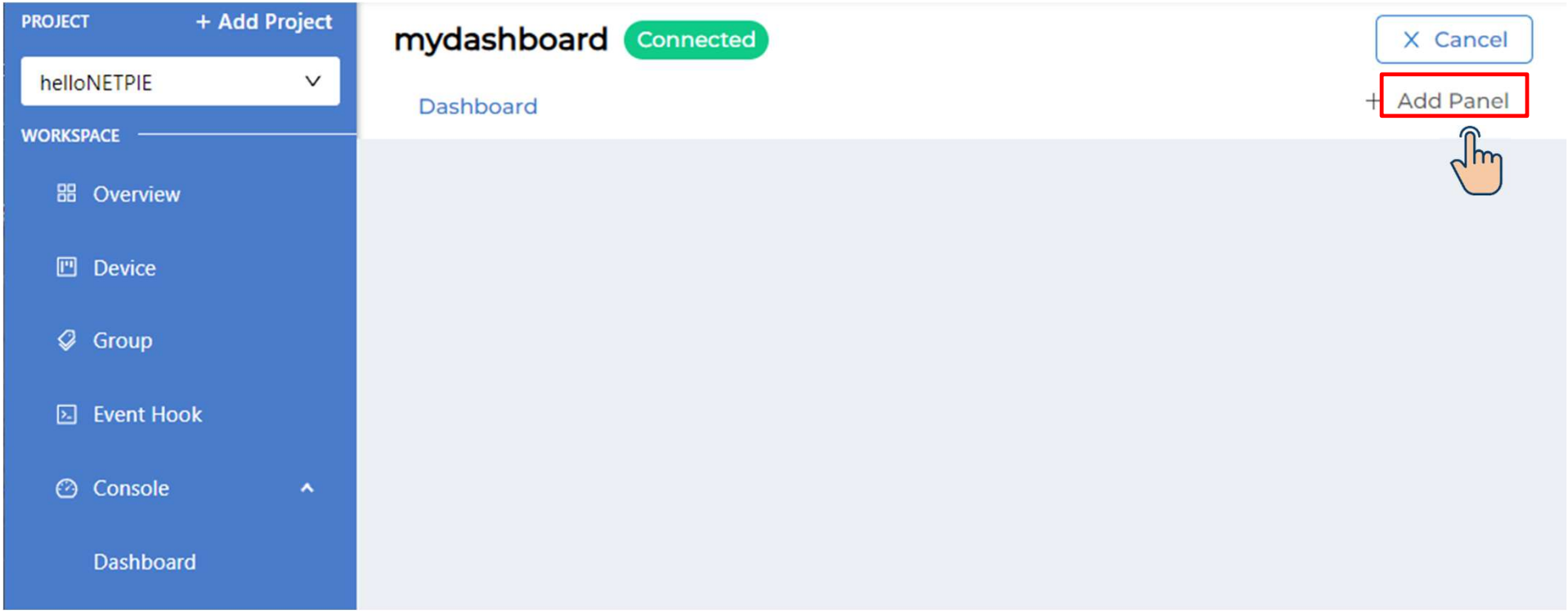
Dashboard

Workshop 6 : Construct a Dashboard on NETPIE2020



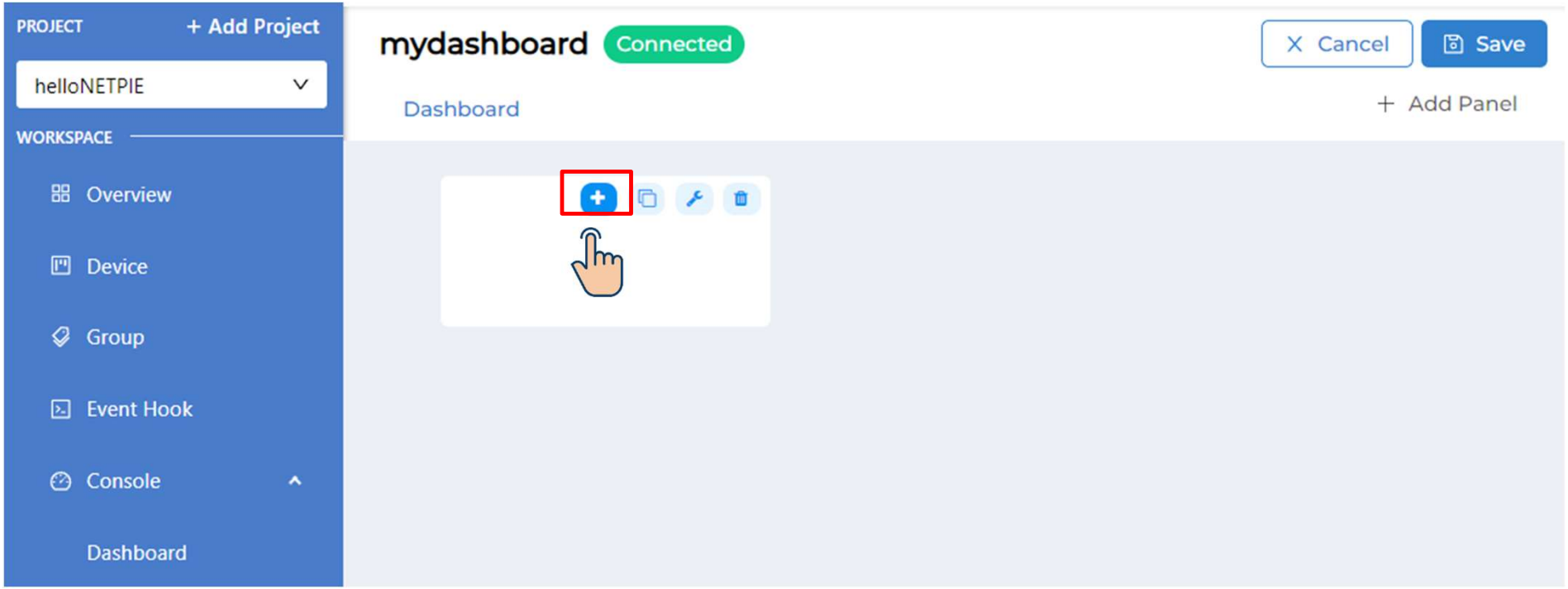
Dashboard

Workshop 6 : Construct a Dashboard on NETPIE2020



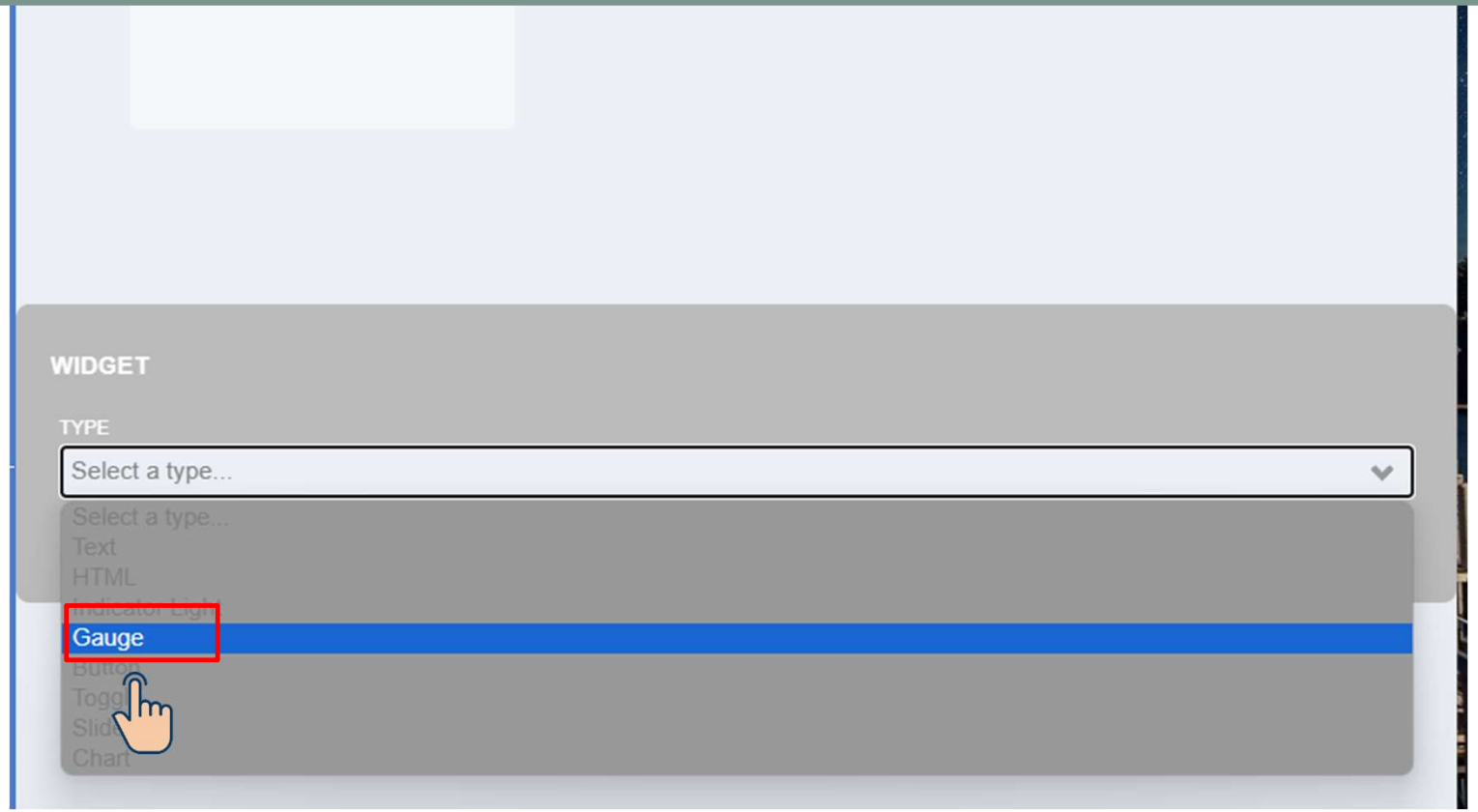
Dashboard

Workshop 6 : Construct a Dashboard on NETPIE2020



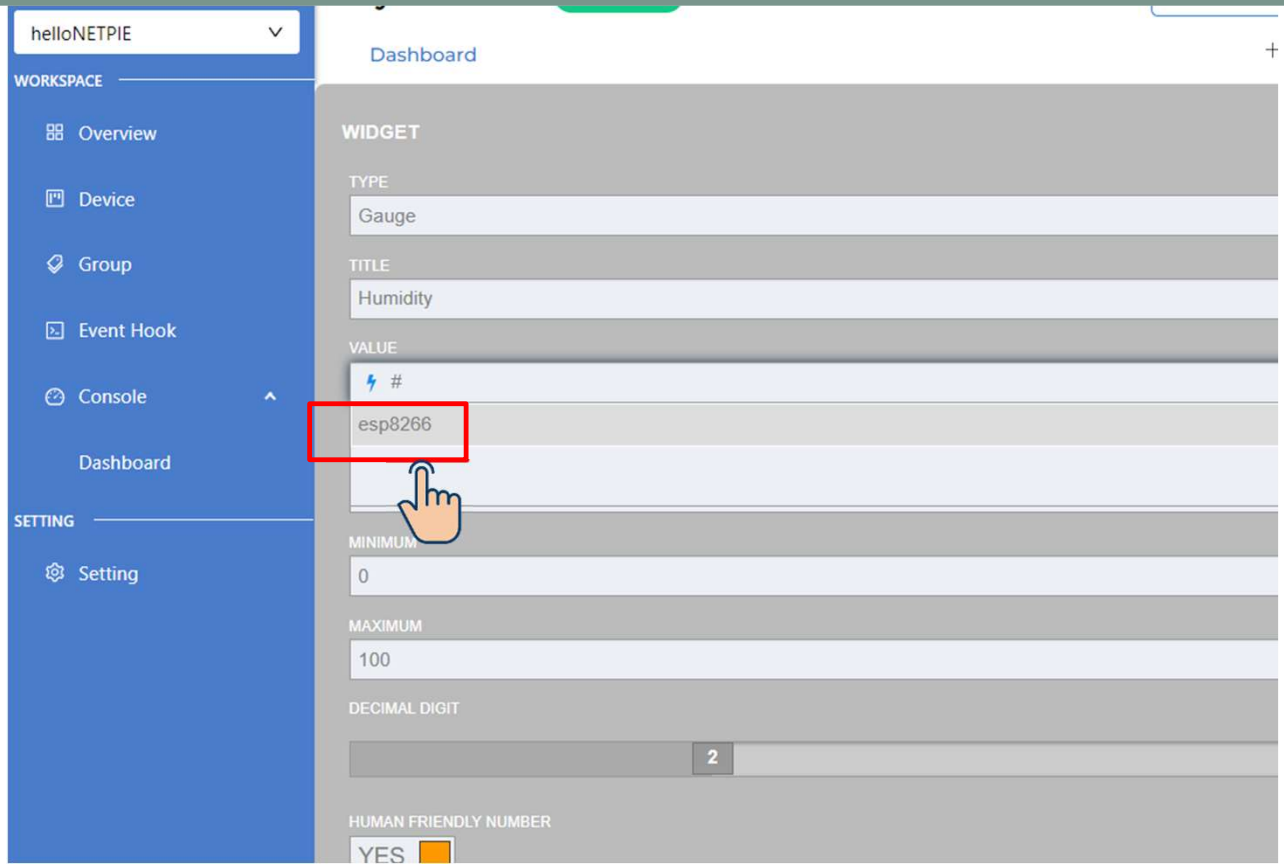
Dashboard

Workshop 6 : Construct a Dashboard on NETPIE2020



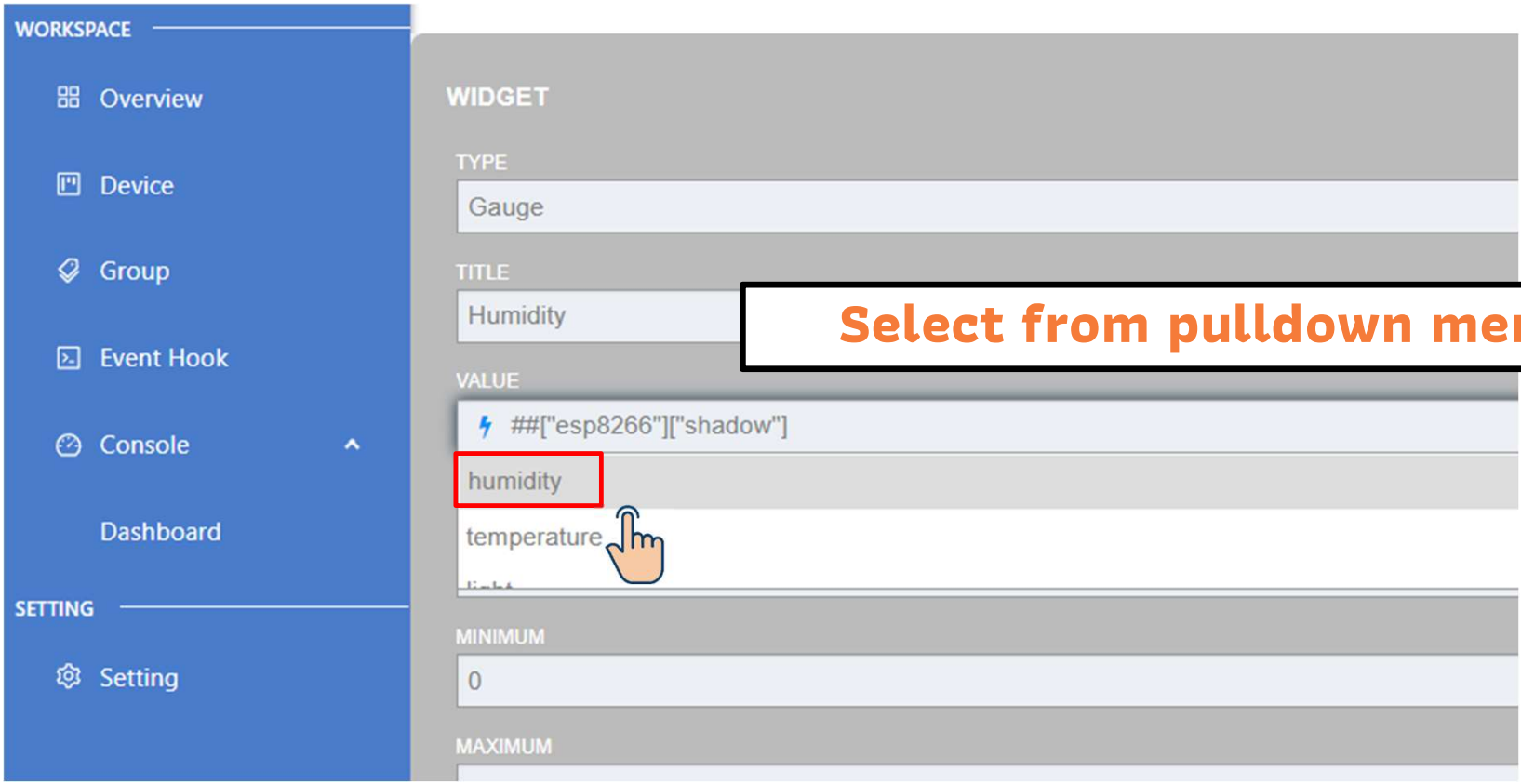
Dashboard

Workshop 6 : Construct a Dashboard on NETPIE2020



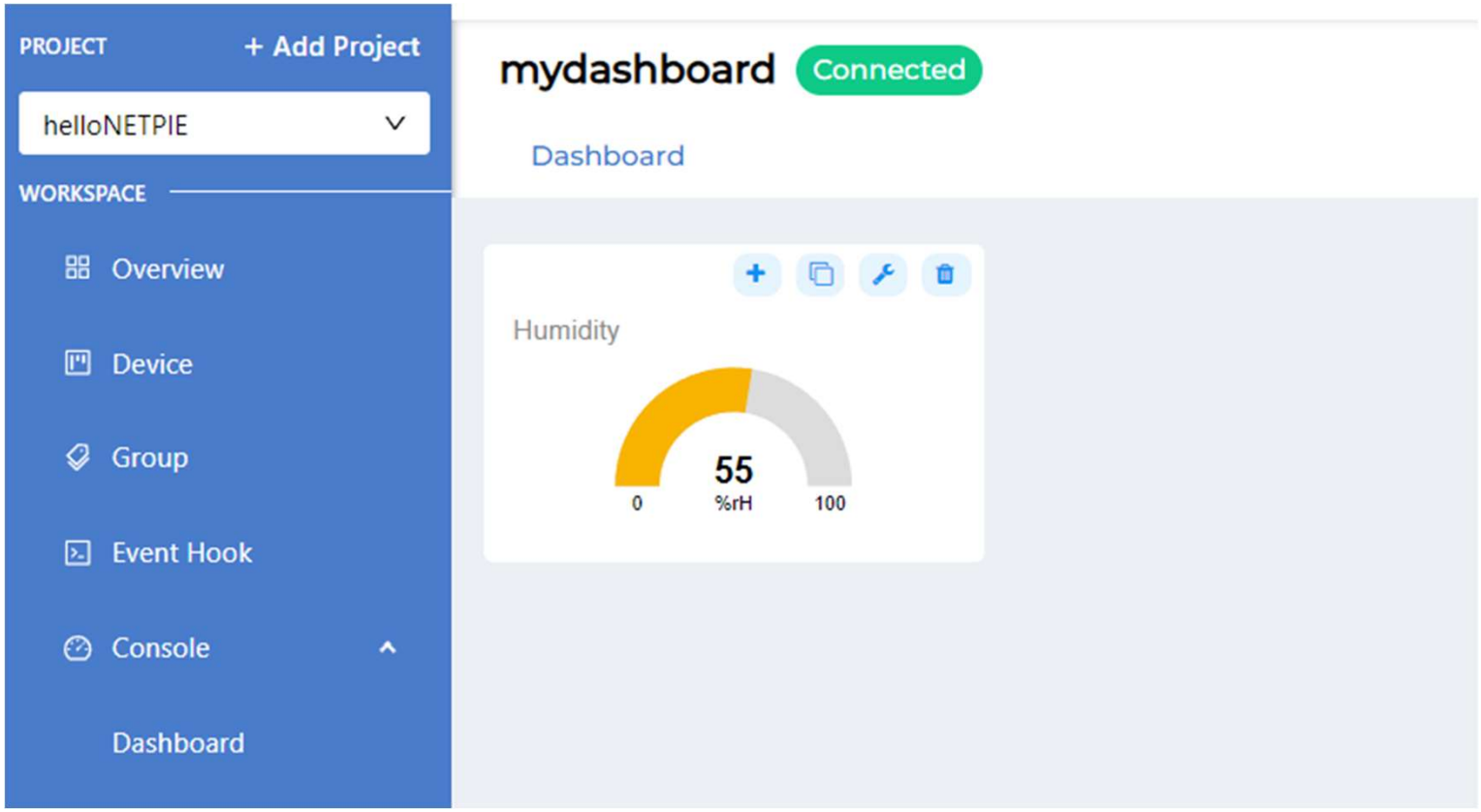
Dashboard

Workshop 6 : Construct a Dashboard on NETPIE2020



Dashboard

Workshop 6 : Construct a Dashboard on NETPIE2020

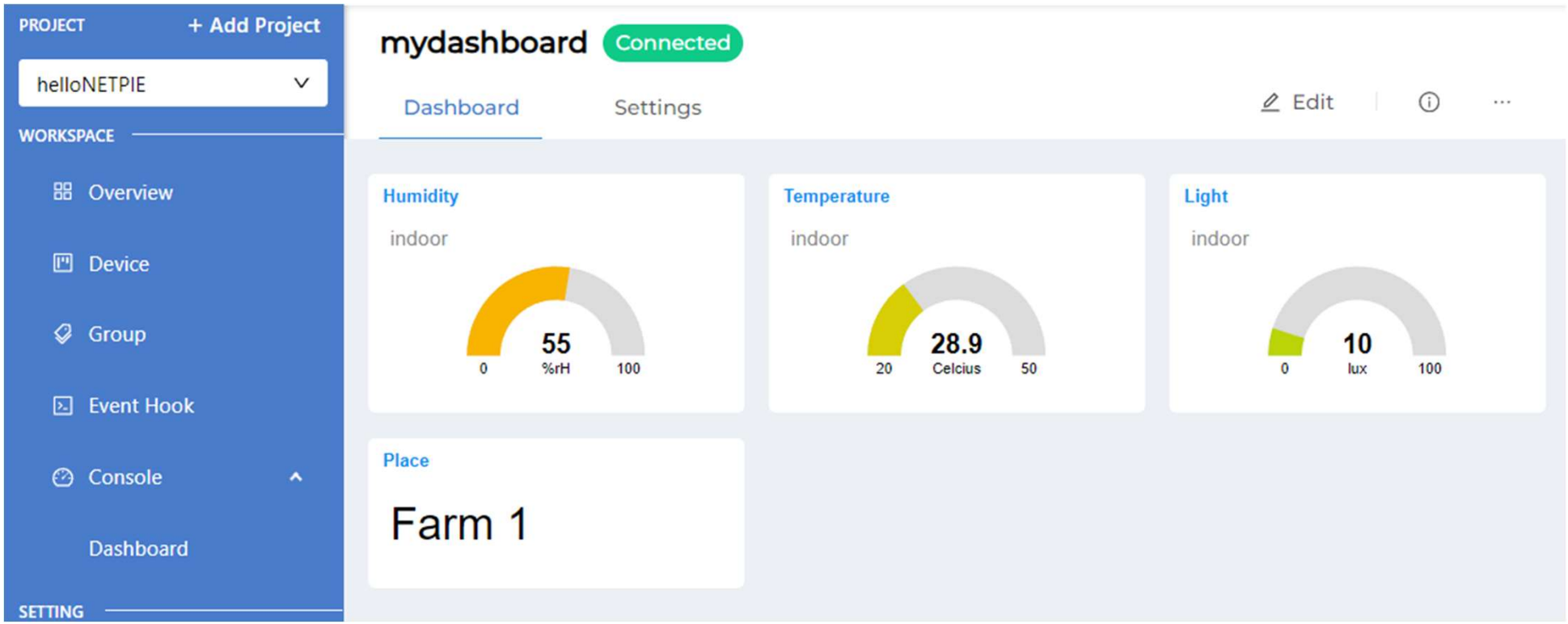


Workshop 6 : Construct a Dashboard on NETPIE2020

The rest of dashboard can be constructed the same way…

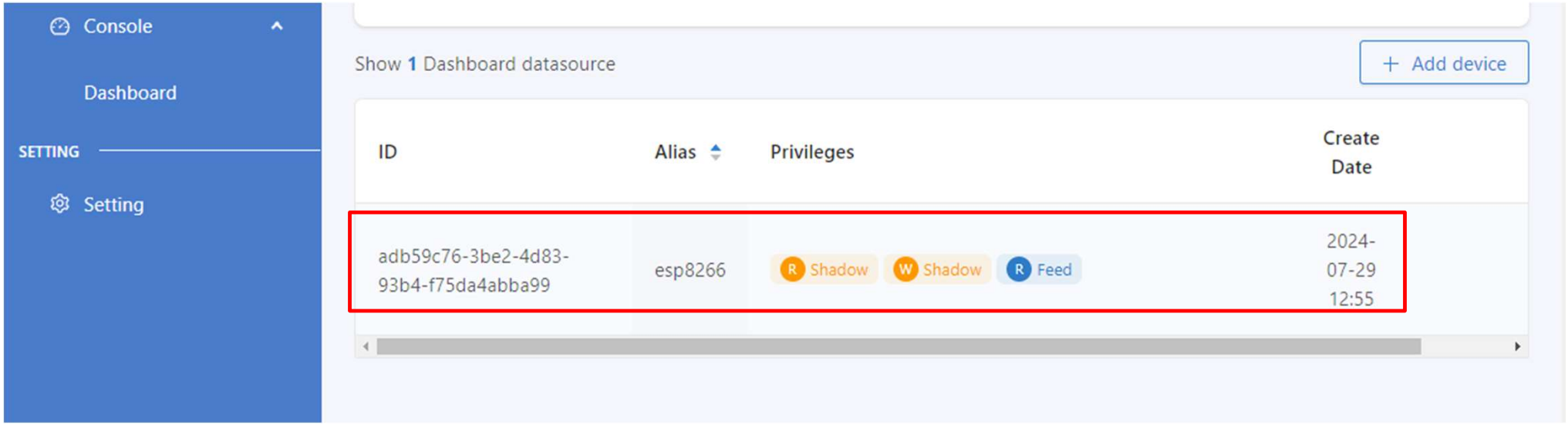
Dashboard

Workshop 6 : Construct a Dashboard on NETPIE2020



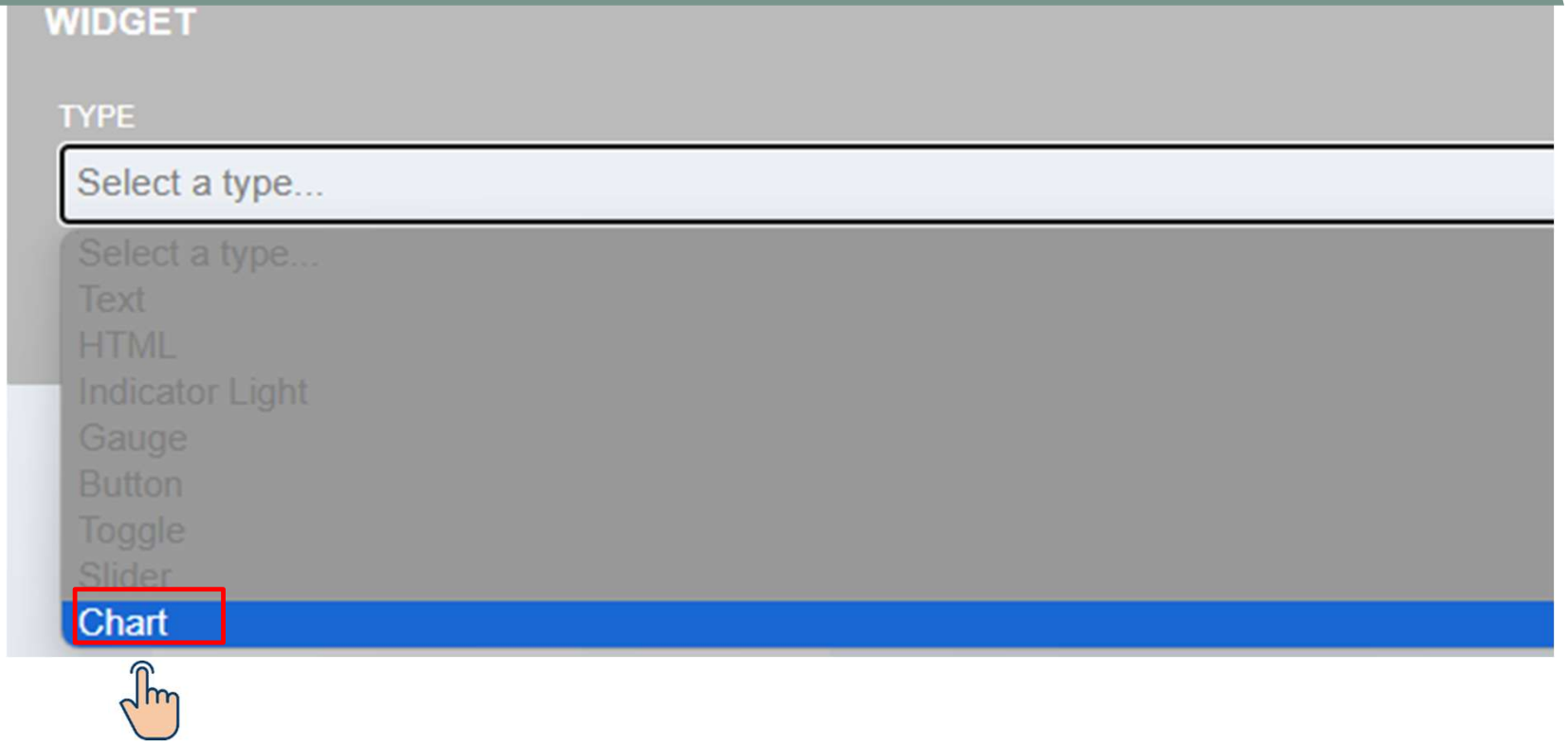
Dashboard

Workshop 6 : Construct a Dashboard on NETPIE2020



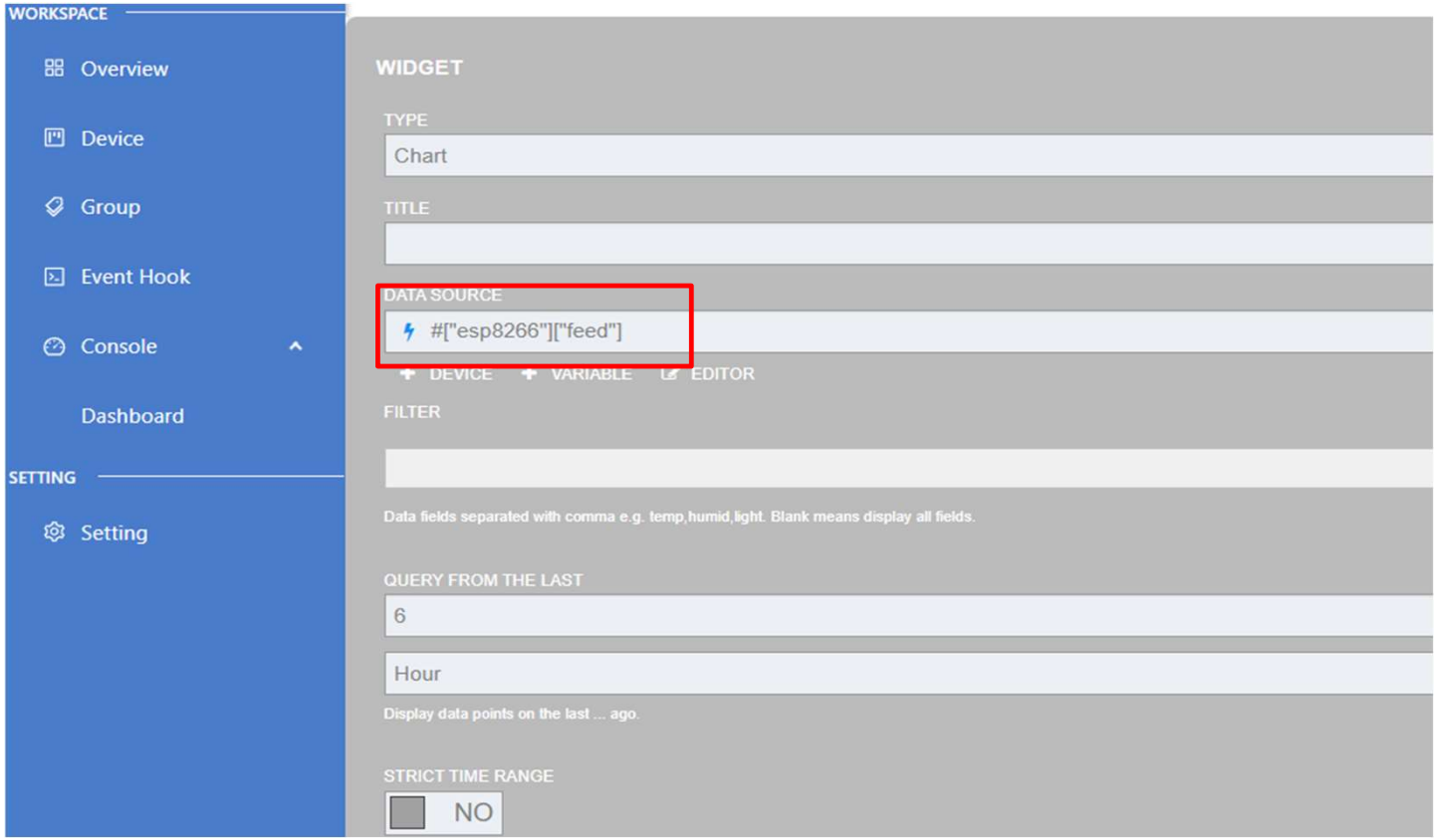
Dashboard

Workshop 6 : Construct a Dashboard on NETPIE2020



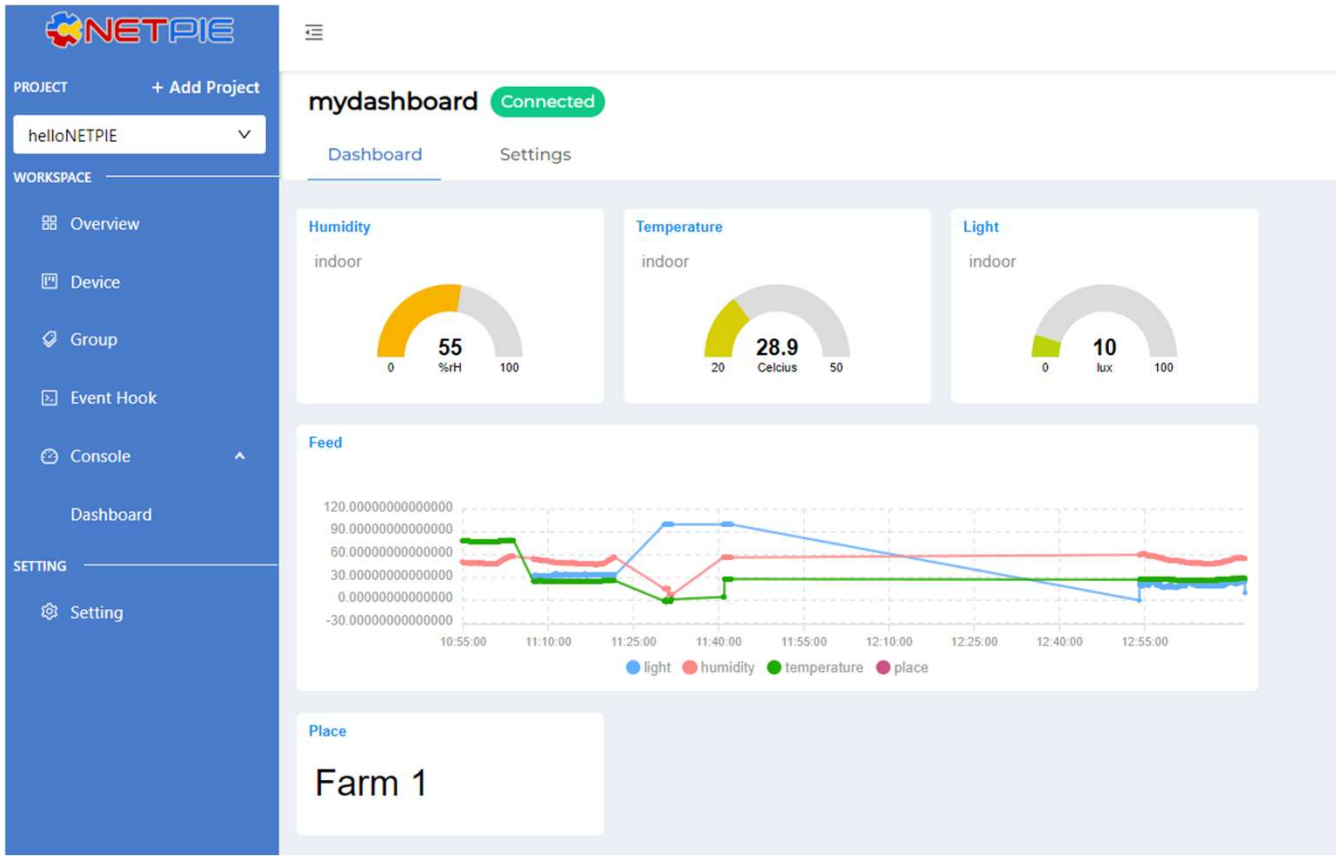
Dashboard

Workshop 6 : Construct a Dashboard on NETPIE2020



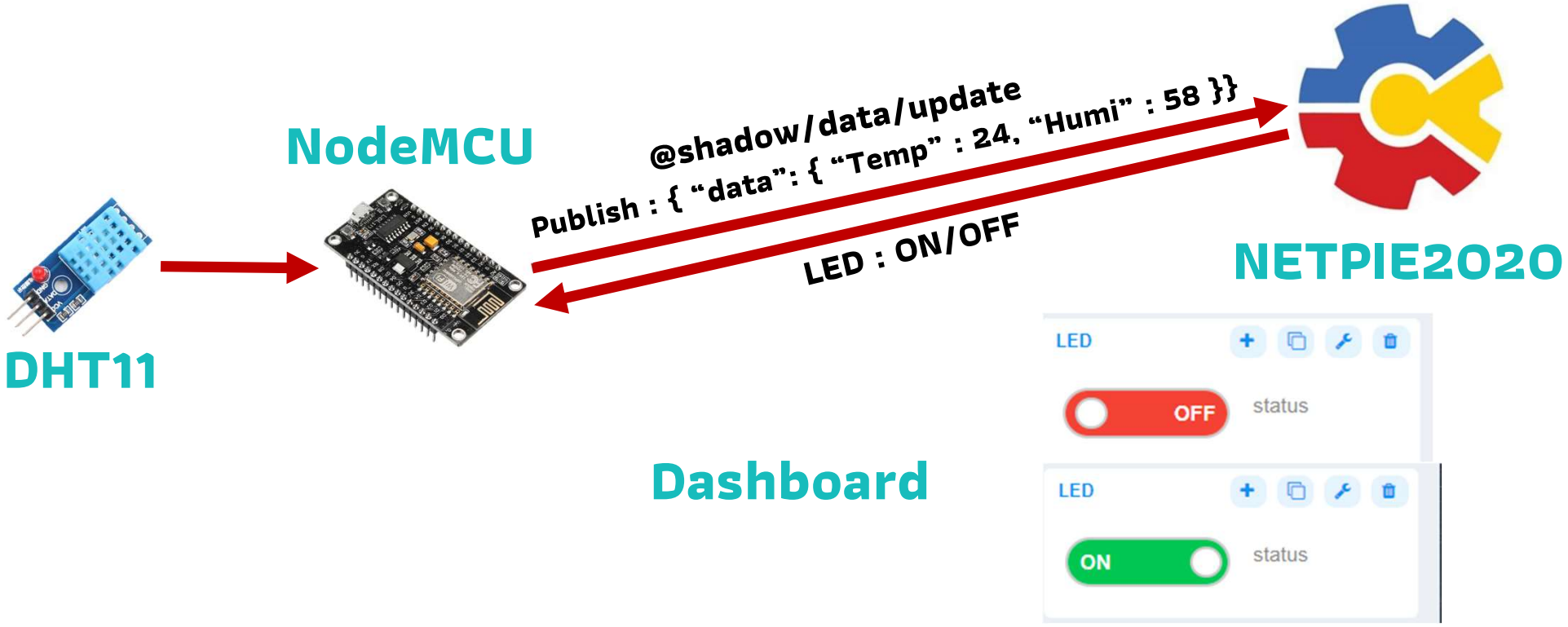
Dashboard

Workshop 6 : Construct a Dashboard on NETPIE2020



Dashboard

Workshop 7 : Control LED on NodeMCU via Dashboard

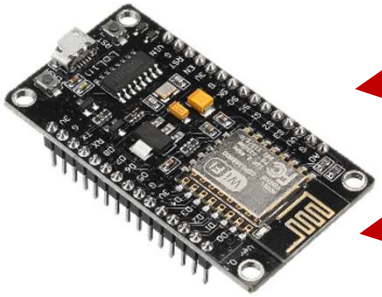


Dashboard

Workshop 7 : Control LEDs on NodeMCU via Dashboard

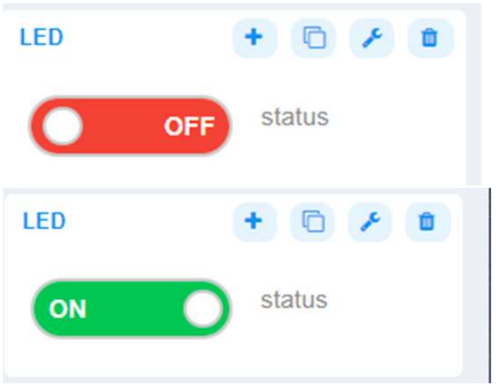
LED on NodeMCU can be controlled via Dashboard by 2 methods

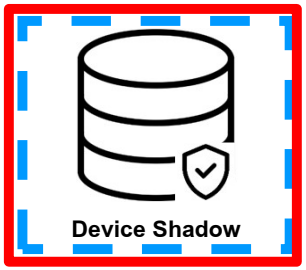
1. Command via Shadow



@shadow/data/update
"led" : "off"

@shadow/data/update
"led" : "oon"





"led" : on

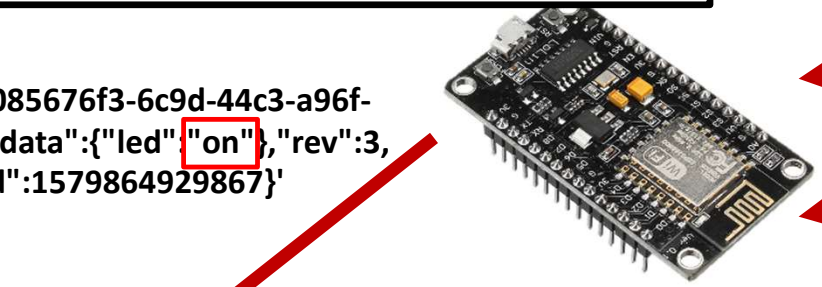
Dashboard

Workshop 7 : Control LEDs on NodeMCU via Dashboard

LED on NodeMCU can be controlled via Dashboard by 2 methods

1. Command via Shadow

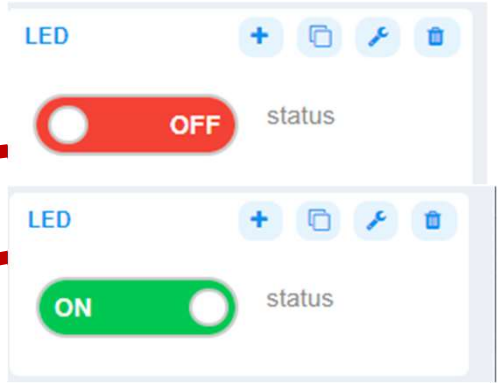
Message sent via shadow has the following format



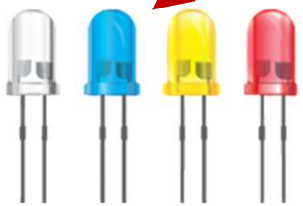
b'{"deviceid":"085676f3-6c9d-44c3-a96f-c94f90412728","data":{"led":"on"},"rev":3, "modified":1579864929867}'

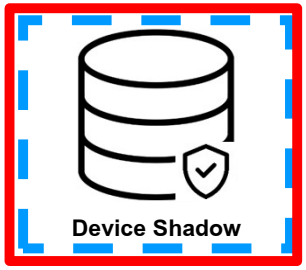
@shadow/data/update
"led" : "off"

@shadow/data/update
"led" : "oon"



On ,Off



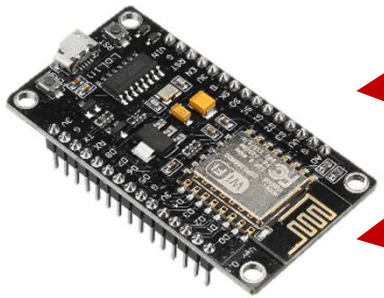


Dashboard

Workshop 7 : Control LEDs on NodeMCU via Dashboard

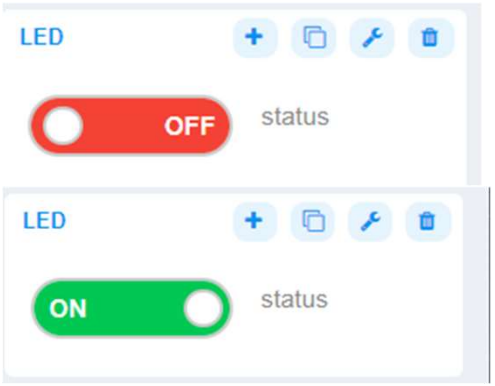
LED on NodeMCU can be controlled via Freeboard by 2 methods

2. Command by Message

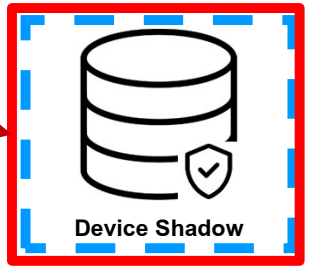


@msg/led
"off"

@msg/led
"on"



@shadow/data/update
"led" : "on"



"led" : "on"

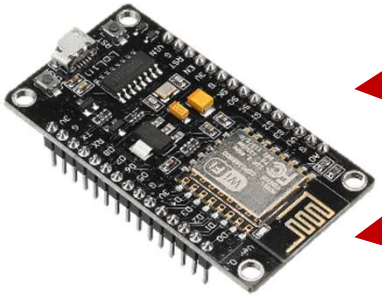
Dashboard

Workshop 7 : Control LEDs on NodeMCU via Dashboard

My preferred method

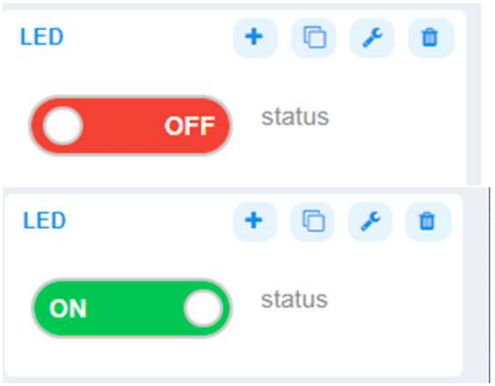
Command = parameter

Formulate command structure

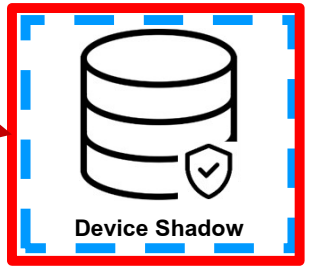


@msg/cmd
"led=off"

@msg/cmd
"led=on"



@shadow/data/update
"led" : "on"



"led" : "on"

Dashboard

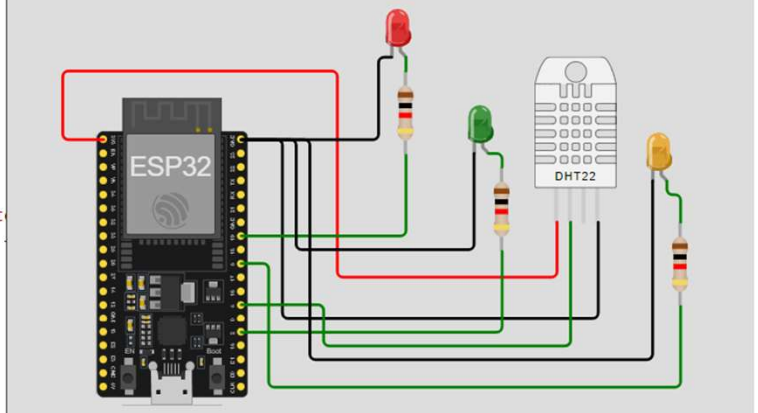
Workshop 7 : Control LEDs on NodeMCU via Dashboard

WOKWi SAVE SHARE Lect6_ws7_cmdparm

sketch.ino | diagram.json | libraries.txt | Library Manager

```
117  void loop() {
118    newDHTRead = millis();
119    if (newDHTRead<lastDHTRead) lastDHTRead = newDHTRead; // prevent overflow
120    if(newDHTRead - lastDHTRead > 2000){
121        lastDHTRead = newDHTRead;
122        float humidity = dht.readHumidity();
123        float temperature = dht.readTemperature();
124        uint16_t light = 100; //LightSensor.GetLightIntensity();
125        String place = "Farm 1";
126          if (!client.connected()) {
127          reconnect();
128        }
129        client.loop();
130        String data = "{\"data\": {\"humidity\":" + String(humidity) + ", \"t
131        + ", \"light\":" + String(light)+ ", \"place\": \"" +  String(place)
132        Serial.println(data);
133        data.toCharArray(msg, (data.length() + 1));
134        client.publish("@shadow/data/update", msg);
135    }
136    while (Serial.available() > 0)   {  // detect new input
137        rcvdstring = Serial.readString();
138        newcmd = 1;
139    }
140    if (newcmd)   { // execute this part only when new command received
141      cmdInt();   // invoke the interpreter
142      newcmd = 0;
143    }
144    delay(100);
145  }
146
```

Simulation



https://wokwi.com/projects/404734188988206081

Workshop 7 : Control LEDs on NodeMCU via Dashboard

Open ws7_cmdparm.ino

Coding in ws7_cmdparm.ino consists of 3 parts

Part 1 Library and variable declaration

1

Variable and LED output pin declaration

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include "DHT.h"
#include <Wire.h>
#include <BH1750FVI.h>

#define LAMP D5
#define VALVE D6
#define FERTILIZER D7

#define DHTPIN D4
#define DHTTYPE DHT11

const char* ssid = "dew2G";
const char* password = "dewninja666";
const char* mqtt_server = "broker.netpie.io";
const int mqtt_port = 1883;
const char* mqtt_Client = "adb59c76-3be2-4d83-93b4-f75da4abba99";
const char* mqtt_username = "A5pGYWRFXjkkqorVLM95Qn4BUbqWbLvf";
const char* mqtt_password = "5Uz6sUqUPiHFPH7gWPP1uFWS6bDyKZiv";

WiFiClient espClient;
PubSubClient client(espClient);
DHT dht(DHTPIN, DHTTYPE);
uint8_t ADDRESSPIN = 15;    // D8, not used
BH1750FVI::eDeviceAddress_t DEVICEADDRESS = BH1750FVI::k_DevAddress_L;
BH1750FVI::eDeviceMode_t DEVICEMODE = BH1750FVI::k_DevModeContHighRes;
```

Workshop 7 : Control LEDs on NodeMCU via Dashboard

2 **Part 2 various functions**

MQTT connection function

Subscribe Topic sent via Dashboard

```
void reconnect() {
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection…");
    if (client.connect(mqtt_Client, mqtt_username,
mqtt_password)) {
      Serial.println("connected");
      // subscribe to command topics
      client.subscribe("@msg/#");
    }
    else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println("try again in 5 seconds");
      delay(5000);
    }
  }
}
```

Workshop 7 : Control LEDs on NodeMCU via Dashboard

2 **Part 2 various functions**

Setup function

```
void setup() {

  pinMode(LAMP, OUTPUT);     // Grow lamp intensity 0 - 100%
  pinMode(VALVE, OUTPUT);     // water valve open 0
  pinMode(FERTILIZER, OUTPUT);     // turn off/on fe
unit
  Serial.begin(115200);
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
```

Assign callback function to MQTT

```
Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
  client.setServer(mqtt_server, mqtt_port);
  client.setCallback(callback);
  dht.begin();
  LightSensor.begin();
}
```

Workshop 7 : Control LEDs on NodeMCU via Dashboard

2 Part 2 callback functions

MQTT message receive function

```
void callback(char* topic, byte* payload, unsigned int length) {
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  String message;
  for (int i = 0; i < length; i++) {
    message = message + (char)payload[i];
  }
  Serial.println(message);
  if(String(topic) == "@msg/cmd") {
    rcvdstring=message;
    cmdInt();  // call command interpreter
  }
}
```

Commands to display MQTT topic and message

If topic is @msg/cmd,
call command intepreter

Workshop 7 : Control LEDs on NodeMCU via Dashboard

3

Part 3 command intepreter

```
void cmdInt(void)
{

    rcvdstring.trim();  // remove leading&trailing whitespace, if any
    // find index of separator ('=')
    sepIndex = rcvdstring.indexOf('=');
    if (sepIndex==-1) {  // no parameter in command
      cmdstring = rcvdstring;
      noparm = 1;
    }
    else  {
    // extract command and parameter
      cmdstring = rcvdstring.substring(0, sepIndex);
      cmdstring.trim();
      parmstring = rcvdstring.substring(sepIndex+1);
      parmstring.trim();
      noparm = 0;
    }
```

Workshop 7 : Control LEDs on NodeMCU via Dashboard

```
if (cmdstring.equalsIgnoreCase("lamp"))   {
  if (noparm==1)   {
    Serial.println("Lamp intensity = "+(String)map(lampvalue,0,1023,0,100) + " %");

  }
  else   {
     parmvalint = parmstring.toInt();
     if (parmvalint > 100) parmvalint = 100; // limit to 100%
     else if (parmvalint<0) parmvalint = 0;
     lamp = parmvalint;
     lampvalue = map(lamp,0,100,0,1023);
     analogWrite(LAMP,lampvalue);
     datastr = "{\"data\": {\"lamp\":" + String(lamp)+"}}";
     datastr.toCharArray(msg, (datastr.length() + 1));
     client.publish("@shadow/data/update", msg);
     Serial.print("Lamp set to ");
     Serial.print(lamp);
     Serial.println(" %");
     Serial.println(datastr);
  }
}
```

3

Part 3 command intepreter

Workshop 7 : Control LEDs on NodeMCU via Dashboard

```
else if (cmdstring.equalsIgnoreCase("valve"))   {
  if (noparm==1)   {
    Serial.println("Valve opening = "+(String)map(valvevalue,0,1023,0,100) + " %");

  }
  else   {
     parmvalint = parmstring.toInt();
     if (parmvalint > 100) parmvalint = 100; // limit to 100%
     else if (parmvalint<0) parmvalint = 0;
     valve = parmvalint;
     valvevalue = map(valve,0,100,0,1023);
     analogWrite(VALVE,valvevalue);
     datastr = "{\"data\": {\"valve\":" + String(valve)+"}}";
     datastr.toCharArray(msg, (datastr.length() + 1));
     client.publish("@shadow/data/update", msg);
     Serial.print("Valve set to ");
     Serial.print(valve);
     Serial.println(" %");
     Serial.println(datastr);
  }
}
```

3

Part 3 command intepreter

Workshop 7 : Control LEDs on NodeMCU via Dashboard

```
  else if (cmdstring.equalsIgnoreCase("fertilizer"))   { // fertilizer feed
   if (noparm==1)   {
     Serial.print("Current fertilizer state = ");
     Serial.println(fertilizer_state);
   }
   else   {
     if (parmstring.equalsIgnoreCase("on")){
       digitalWrite(FERTILIZER,1);
       client.publish("@shadow/data/update", "{\"data\" : {\"fertilizer\" : \"on\"}}");
       Serial.println("FERTILIZER ON");
       fertilizer_state = parmstring;
     }
     else if (parmstring.equalsIgnoreCase("off")) {
       digitalWrite(FERTILIZER,0);
       client.publish("@shadow/data/update", "{\"data\" : {\"fertilizer\" : \"off\"}}");
       Serial.println("FERTILIZER OFF");
       fertilizer_state = parmstring;
     }
     else Serial.println("Invalid fertilizer state");
   }
}
```

3

Part 3 command intepreter

Workshop 7 : Control LED on NodeMCU via Freeboard

3 **Part 3 loop function**

Use millis() for timing instead of delay() to avoid blocking the execution while waiting Freeboard message

```
void loop() {
  newDHTRead = millis();
  if (newDHTRead<lastDHTRead) lastDHTRead = newDHTRead; // prevent overflow
  if(newDHTRead - lastDHTRead > 2000){
      lastDHTRead = newDHTRead;
      float humidity = dht.readHumidity();
      float temperature = dht.readTemperature();
      uint16_t light = LightSensor.GetLightIntensity();
      String place = "Farm 1";
        if (!client.connected()) {
        reconnect();
      }
      client.loop();
      String data = "{\"data\": {\"humidity\":" + String(humidity) + ", \"temperature\":" + String(temperature)
      + ", \"light\":" + String(light)+ ", \"place\": \"" +  String(place) + "\"}}";
      Serial.println(data);
      data.toCharArray(msg, (data.length() + 1));
      client.publish("@shadow/data/update", msg);
  }
```

Dashboard

Workshop 7 : Control LED on NodeMCU via Dashboard

Device Schema in JSON format

```
{
 "additionalProperties": false,
 "properties": {
  "humidity": {
   "operation": {
    "store": {
     "ttl": "7d"
    },
    "transform": {
     "expression": "[{$.temperature]*1.8] + 32"
    }
   },
   "type": "number"
  },
  "place": {
   "operation": {
    "store": {
     "ttl": "7d"
    }
   },
   "type": "string"
  },
  "lamp": {
   "operation": {
    "store": {
     "ttl": "7d"
    }
   },
   "type": "string"
  }
 }
}
```

Maybe easier to set additionalProperties to true

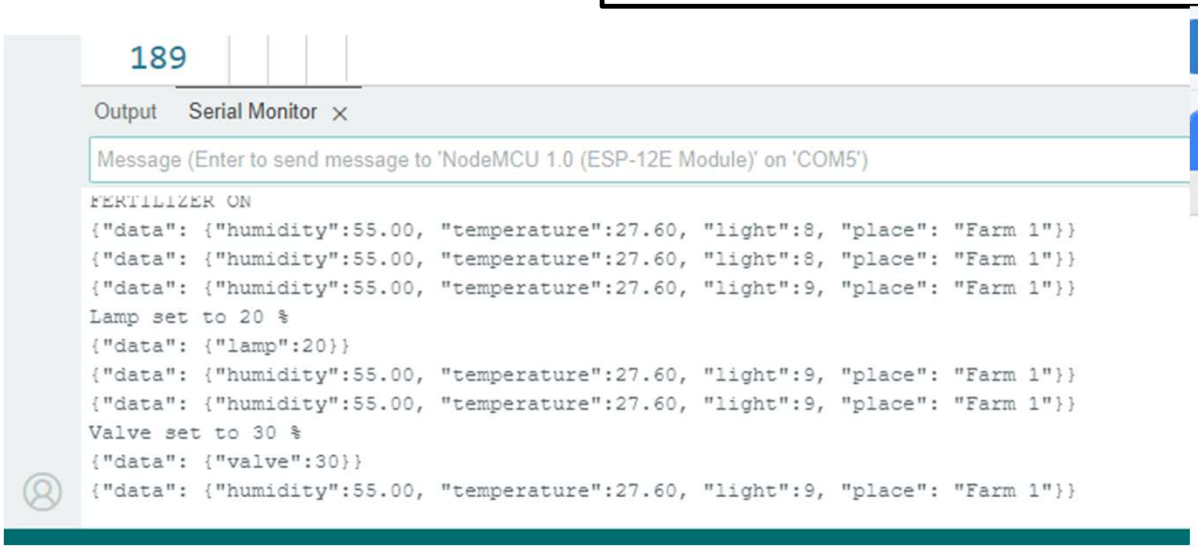
Add code on Schema for lamp, valve and fertilizer

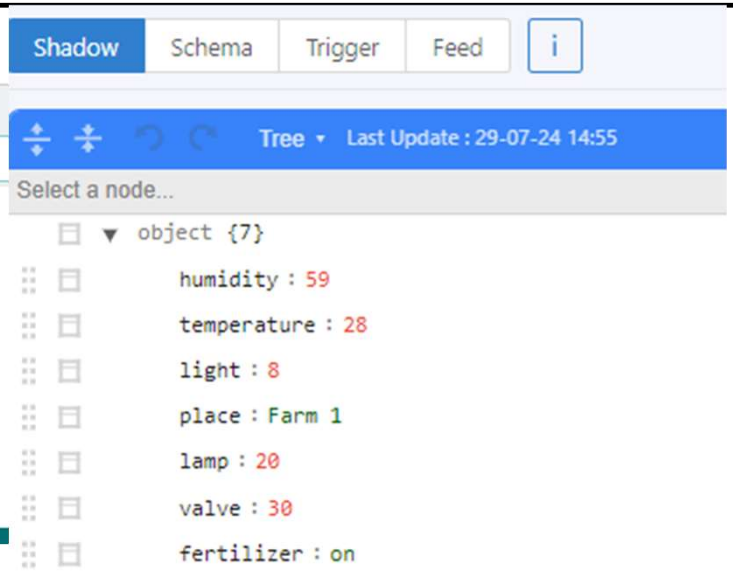
Record **lamp** data with property
- Store for 7 days
- Variable type is number

Dashboard

Workshop 7 : Control LEDs on NodeMCU Dashboard

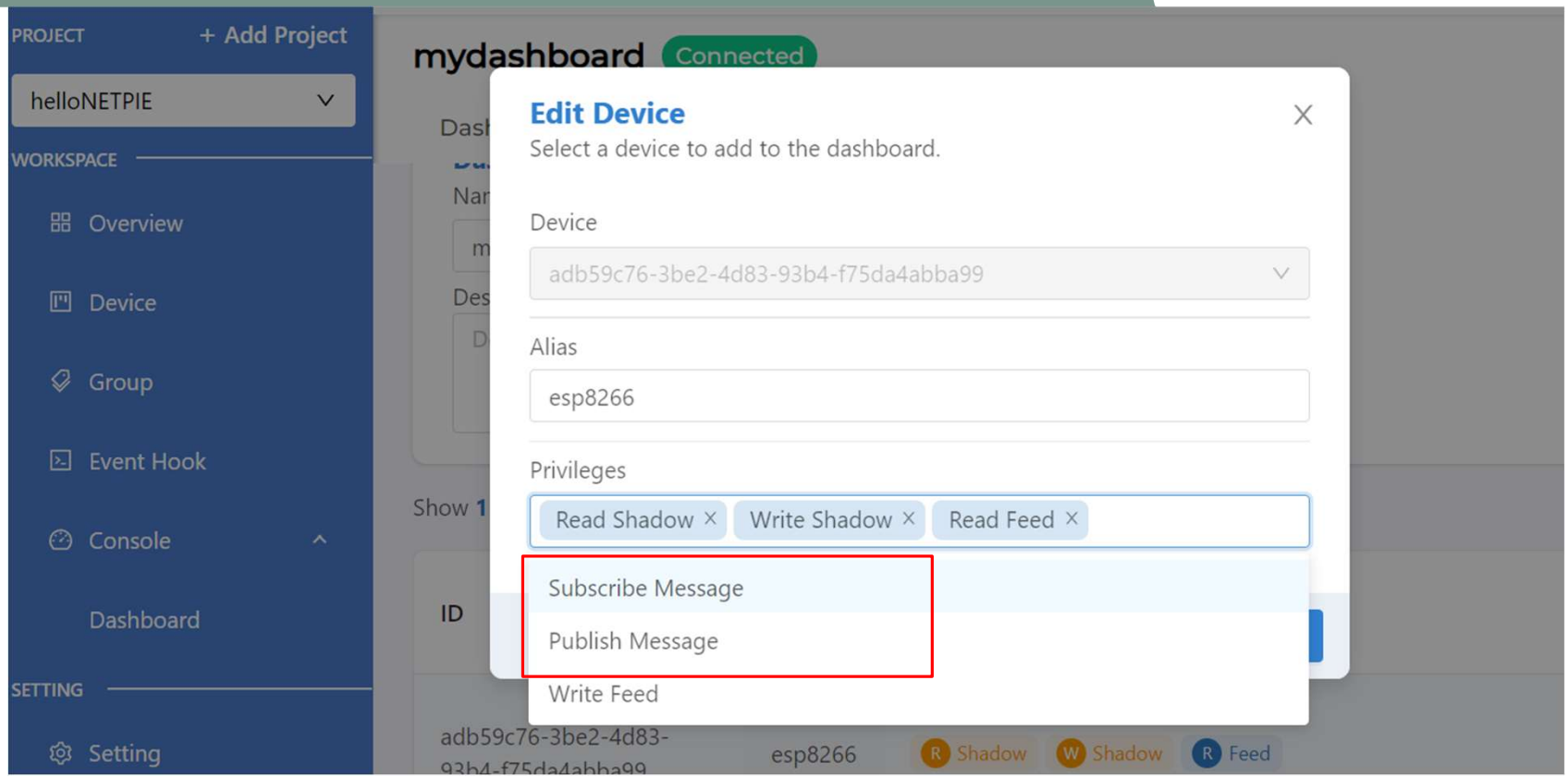
Issue serial port commands to write lamp, Valve, fertilizer on shadow. Ex. lamp=20





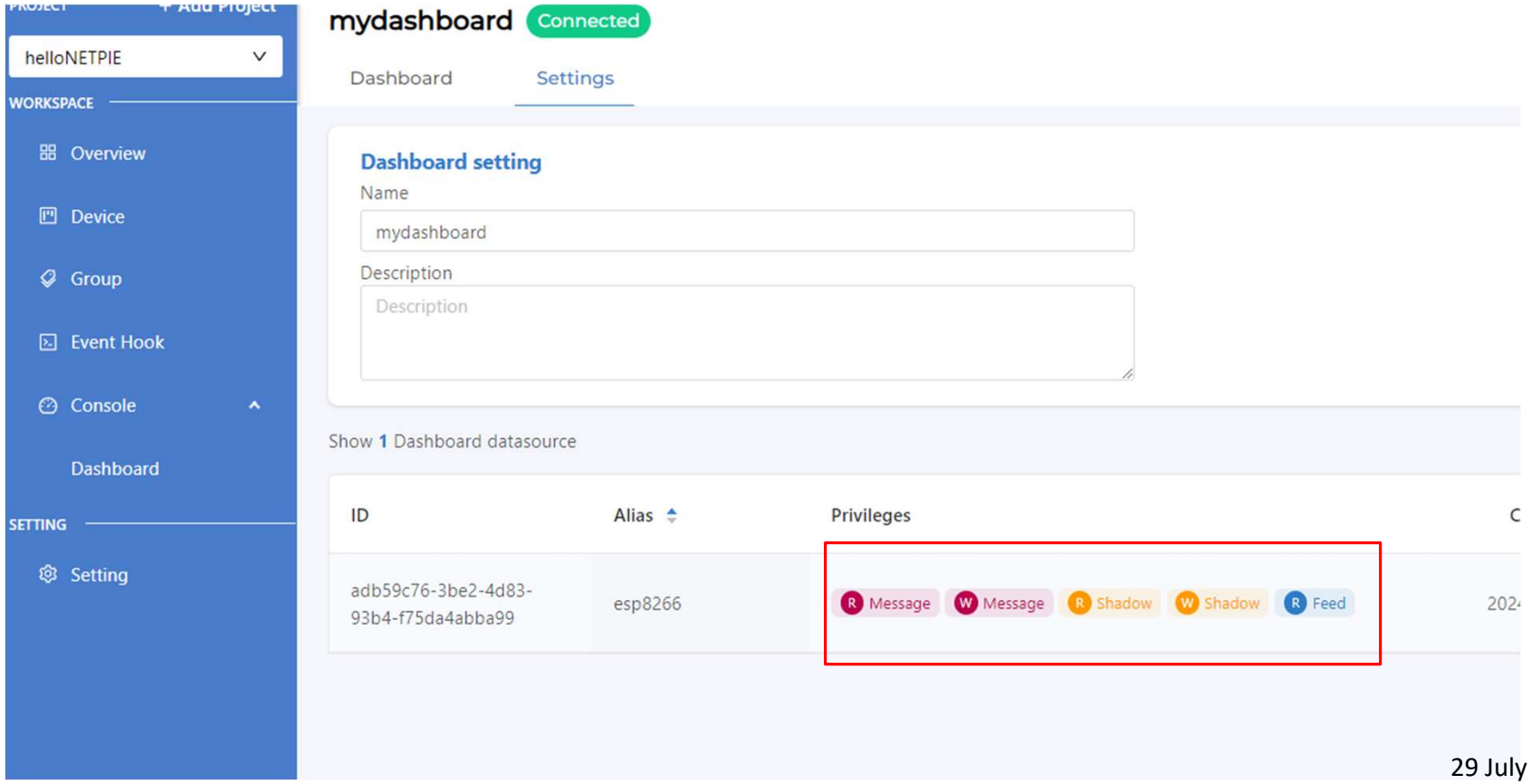
Dashboard

Workshop 7 : Control LEDs on NodeMCU Dashboard



Dashboard

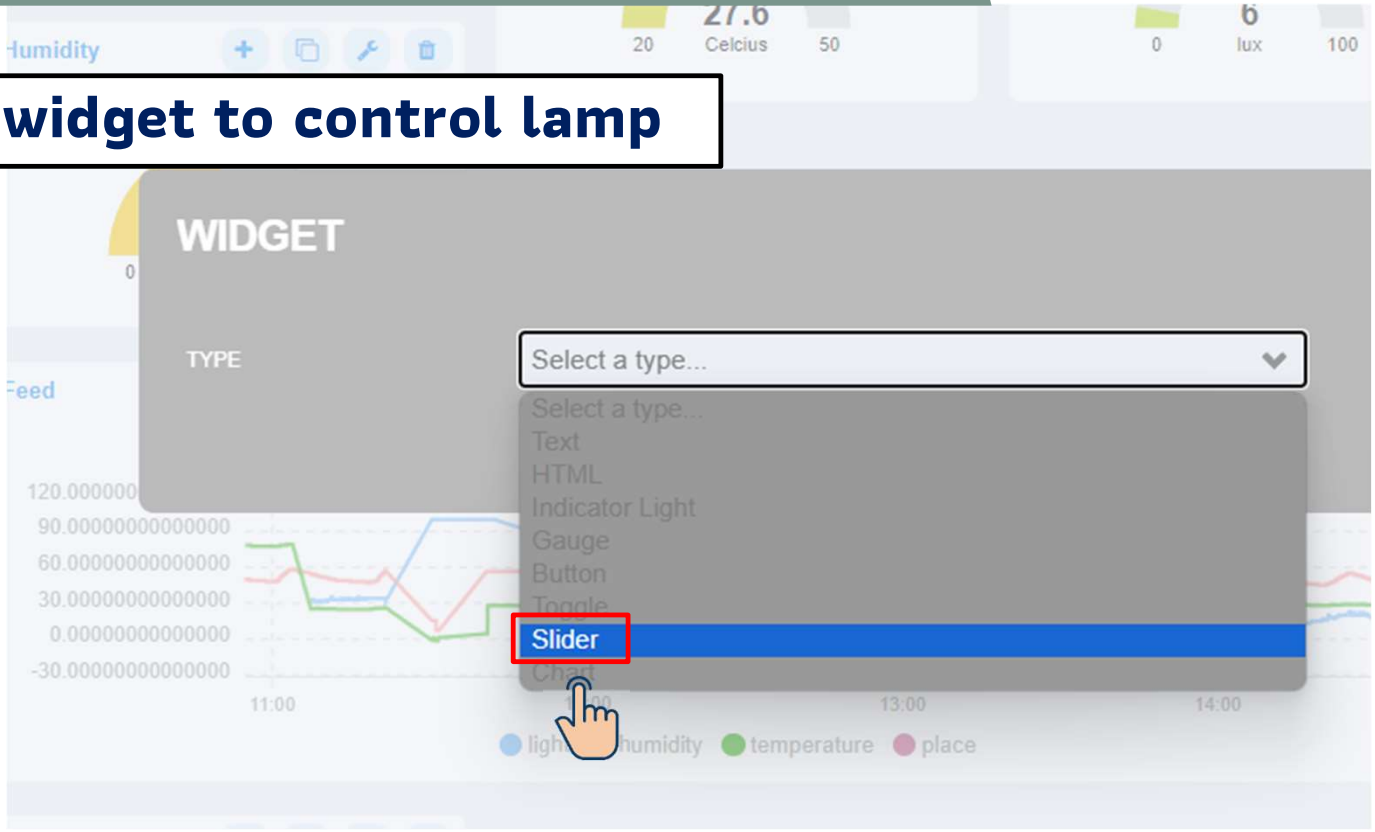
Workshop 7 : Control LEDs on NodeMCU Dashboard



Dashboard

Workshop 7 : Control LEDs on NodeMCU Dashboard

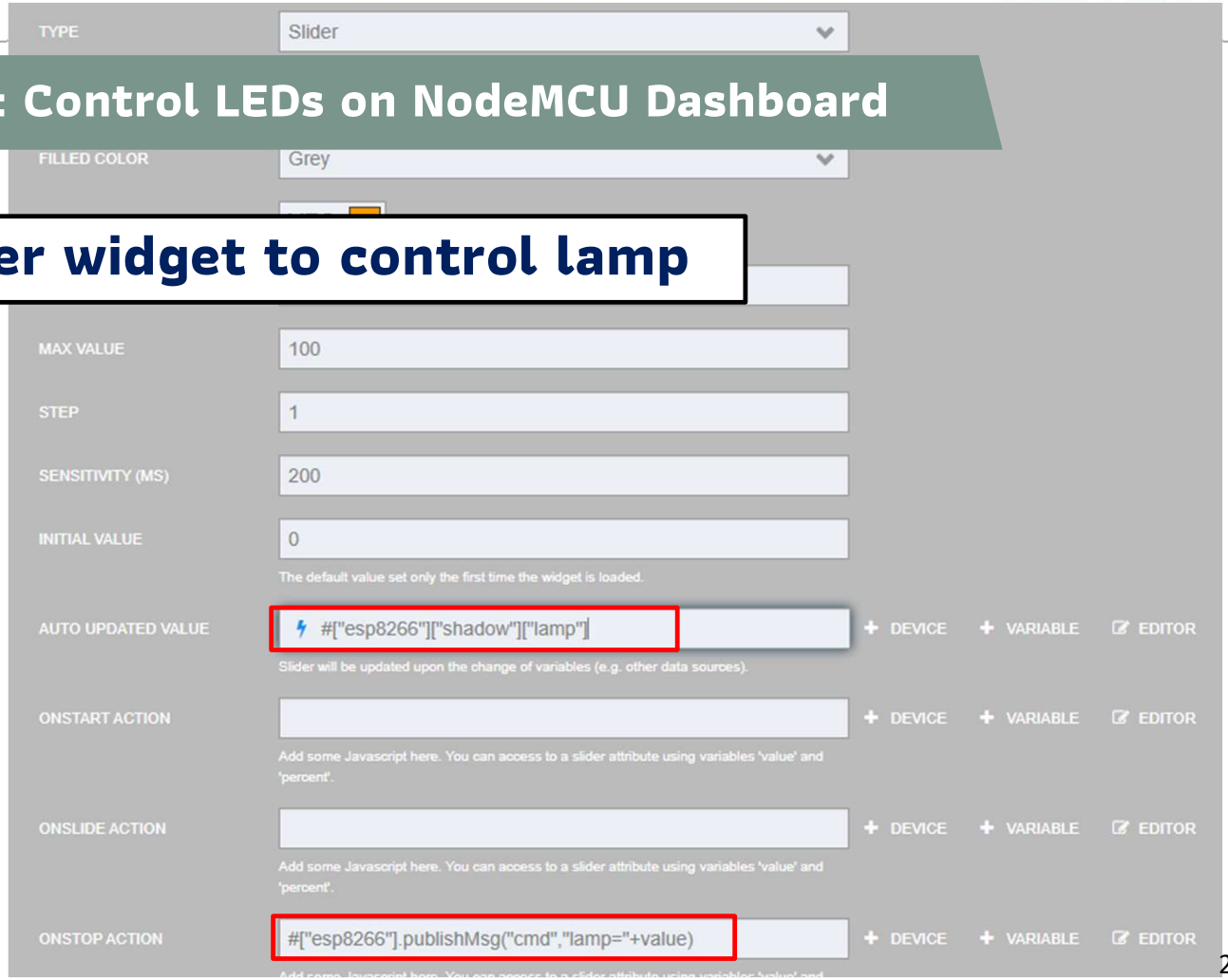
Create a slider widget to control lamp



Dashboard

Workshop 7 : Control LEDs on NodeMCU Dashboard

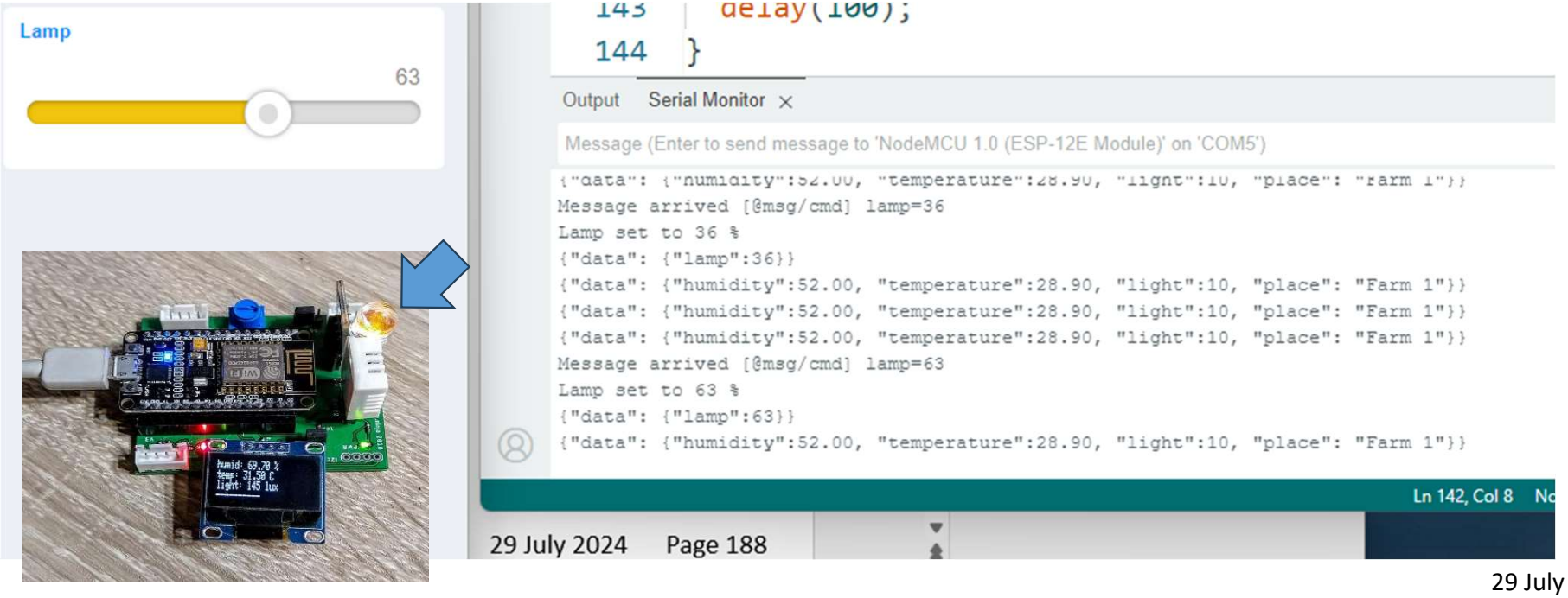
Create a slider widget to control lamp



Dashboard

Workshop 7 : Control LEDs on NodeMCU Dashboard

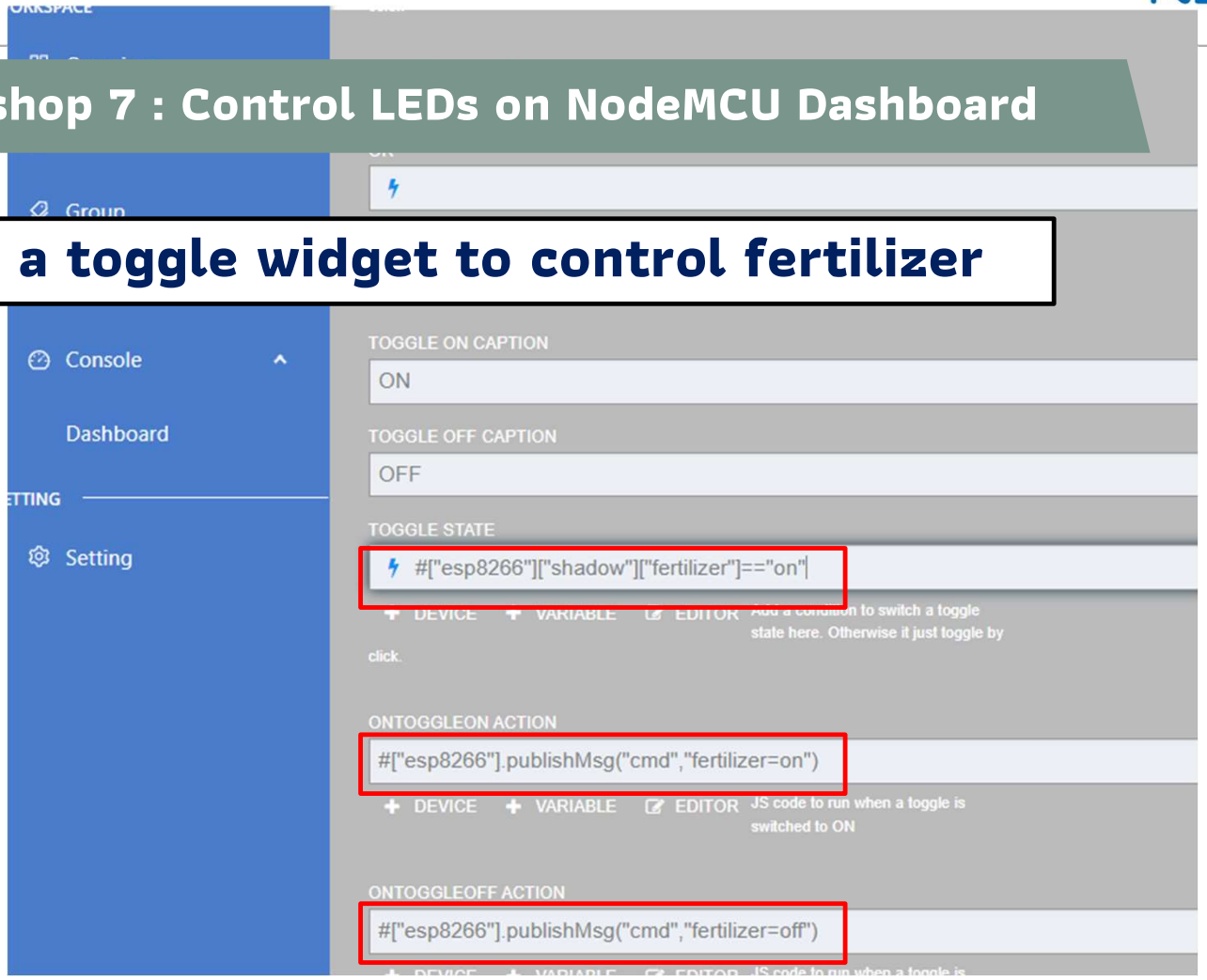
Create a slider widget to control lamp



Dashboard

Workshop 7 : Control LEDs on NodeMCU Dashboard

Create a toggle widget to control fertilizer



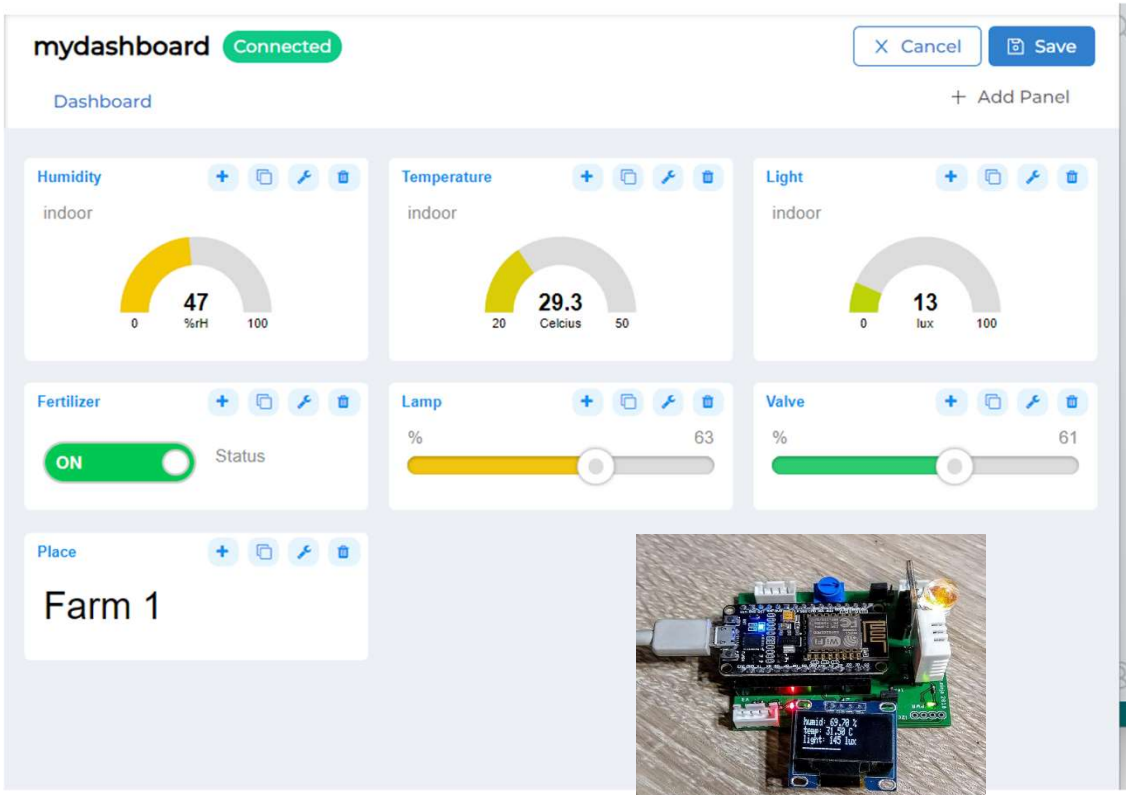
Dashboard

Workshop 7 : Control LEDs on NodeMCU Dashboard

It is left as an exercise to create another slider for valve [represented by a small green LED].

Dashboard

Workshop 7 : Control LEDs on NodeMCU Dashboard




```
134   }
135   while (Serial.available() > 0)   {
136     rcvdstring = Serial.readString();
137     newcmd = 1;
138   }
139   if (newcmd)   { // execute this par
140     cmdInt();   // invoke the interpr
141     newcmd = 0;
142   }
143   delay(100);
144 }
```

Output | Serial Monitor

Message (Enter to send message to 'NodeMCU 1.0 (ESP-12E Module)' on 'COM5')

```
Message arrived [@msg/cmd] fertilizer=off
FERTILIZER OFF
{"data": {"humidity":48.00, "temperature":29.30, "light":12, "place":
Message arrived [@msg/cmd] fertilizer=on
FERTILIZER ON
{"data": {"humidity":47.00, "temperature":29.30, "light":13, "place":
{"data": {"humidity":47.00, "temperature":29.30, "light":13, "place":
Message arrived [@msg/cmd] lamp=63
Lamp set to 63 %
{"data": {"lamp":63}}
{"data": {"humidity":47.00, "temperature":29.30, "light":13, "place":
```

Note for the rest of this Powerpoint file

Since the rest of this presentation is not covered in my course.
The original slides are left intact.

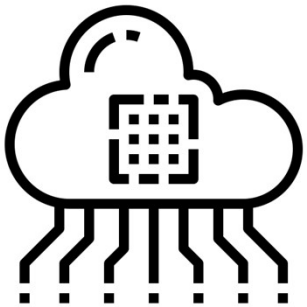
RESTful API

RESTful API

What is RESTful API?

It is a channel for the device to call the Platform service via the RESTful API which uses HTTP protocol, suitable for integrating various systems, either existing or new development, without limitation by programming language or hardware. There are two groups of APIs:

1. Device API
2. Data Store API



Module 5 : Restful API

Device API

It is an API related to device management or calling.
The EndPoint is https://api.netpie.io/v2/device

Publish message to a topic

```
curl -X PUT "https://api.netpie.io/v2/device/message?topic=%40msg%2Ftest" -H "Authorization: Basic ClientID:Token" -H "Content-Type: text/plain" -d "message"
```

Read Shadow Data of Device

```
curl -X GET "https://api.netpie.io/v2/device/shadow/data" -H "Authorization: Basic ClientID:Token"
```

Write Shadow Data using Merge method

```
curl -X PUT "https://api.netpie.io/v2/device/shadow/data" -d "{data:{humid:63.7, temp:25.2}}" –H "Authorization: Device ClientID:Token"
```

Module 5 : Restful API

Device API

Write Shadow Data using Overwrite method

```
curl -X POST "https://api.netpie.io/v2/shadow/data" -d "{data:{temp:31.7}}" -H
"Authorization: Device ClientID:Token"
```

Write to Shadow with specified time in the past

```
curl -X PUT " https://api.netpie.io/v2/shadow/data" -d "{data:{humid:61.9,
temp:28.6,timestamp:1566863843262}}" -H "Authorization: Device ClientID:Token"
```

Module 5 : Restful API

Data Store API

It is an API that deals with retrieving data stored in Time series Database, where the Domain name of API is https//api.netpie.io/v2/feed . The databse is stored in KairosDB format, so querying the parameters can be done using the same format as KairosDB

Read data from Timeseries Database of Device

```
curl -X POST "https://api.netpie.io/v2/feed/api/v1/datapoints/query" –H "Content-Type: application/json" -H "Authorization: BeareruserToken" -d ' {
{\"start_relative\":{\"value\":1,\"unit\":\"days\"},\"metrics\":[{\"name\":\"ClientID\",\"tags\":{\"attr\":[\"temperature\"]},\"limit\":50,\"group_by\":[{\"name\":\"tag\",\"tags\":[\"attr\"]}],\"aggregators\":[{\"name\":\"avg\",\"sampling\":{\"value\":1,\"unit\":\"minutes\"}}]}]}"
```

Webpage to Generate Code for API calls

https://trial-api.netpie.io/

NETPIE2020 Quota

NETPIE2020 Quota

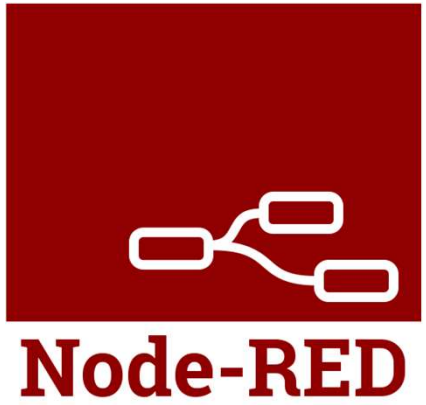
NETPIE2020 Quota

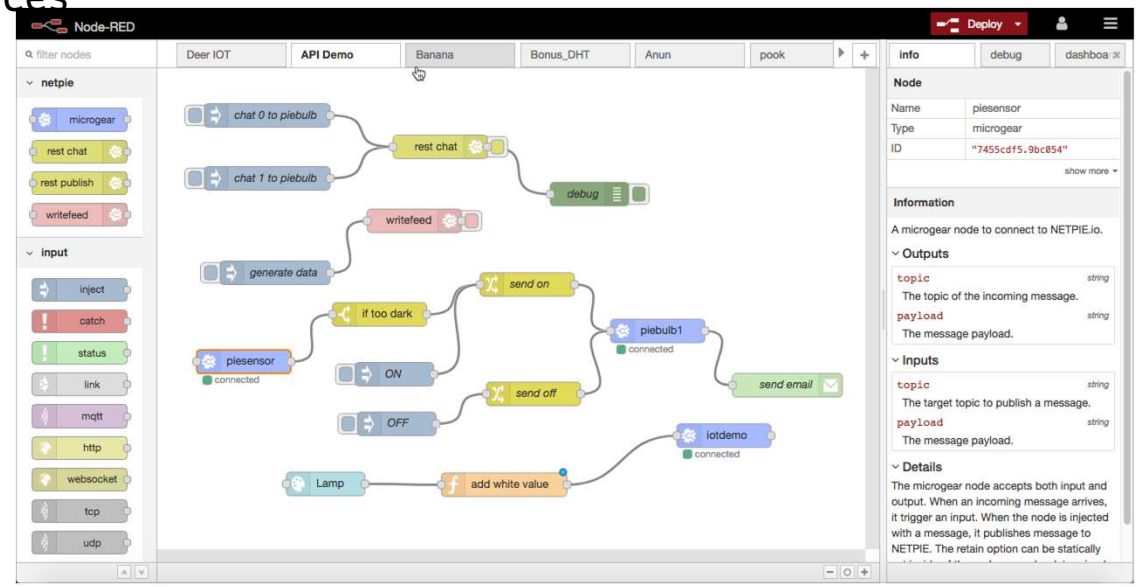
Connecting limit	10 Devices
Project	3 Projects
Real time message	9,000,000 Messages/Month
Data Storage	1,000,000 Points-Month
Shadow read/write	500,000 Operations/Month
API Call	800,000 Operations/Month
Trigger and Action	5,000 Operations/Month
Dashboard	3 Freeboards/Project
Freeboard Connection	3 Concurrent Views

Node-RED

What is Node-RED ?

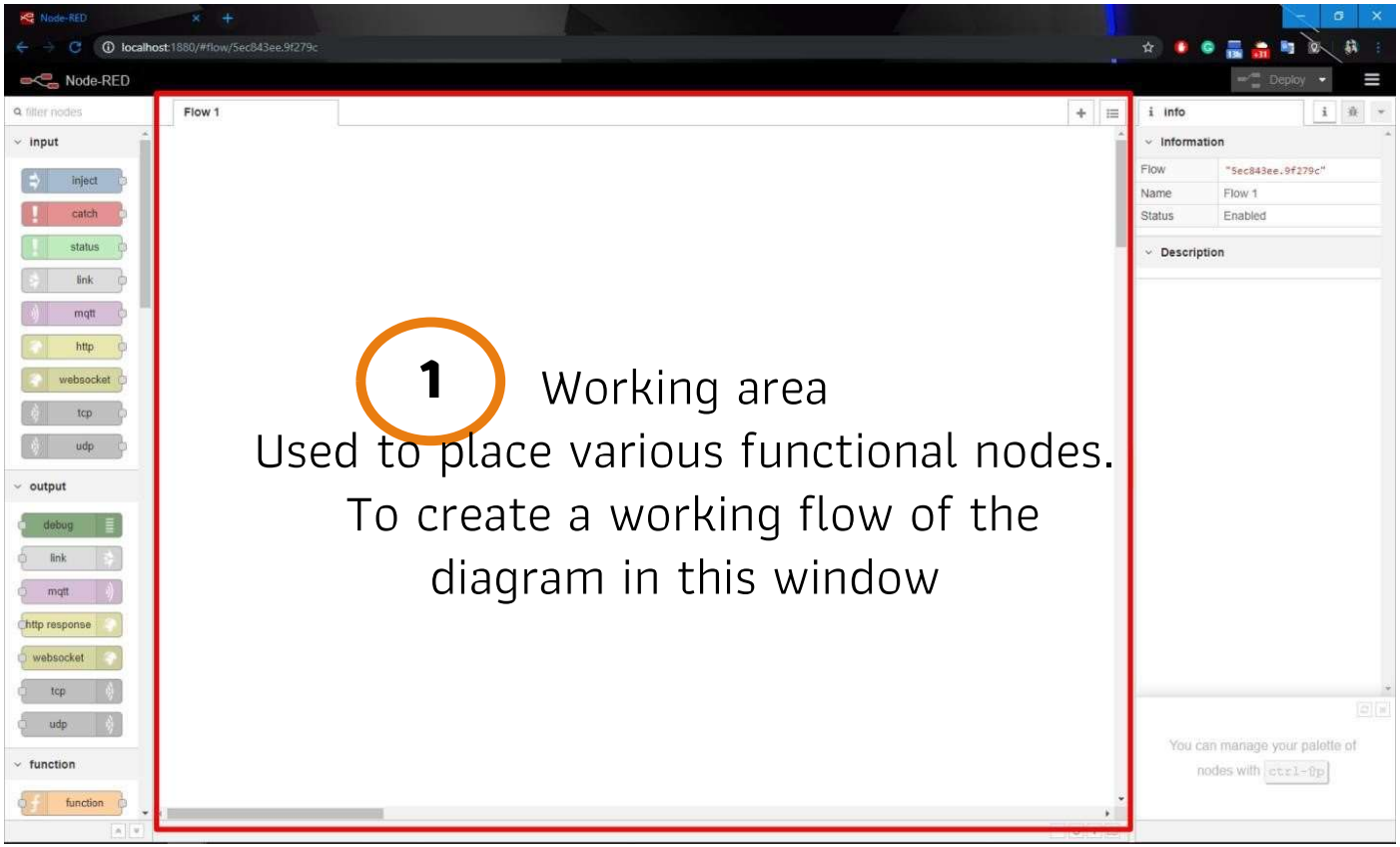
Node-RED is a tool for programmers to connect their hardware devices to APIs [Application Programming Interface] and online services in new ways. They're available for free as well as for paid portions. Node-RED can design APIs. To receive, calculate, convert, store, or connect to other services





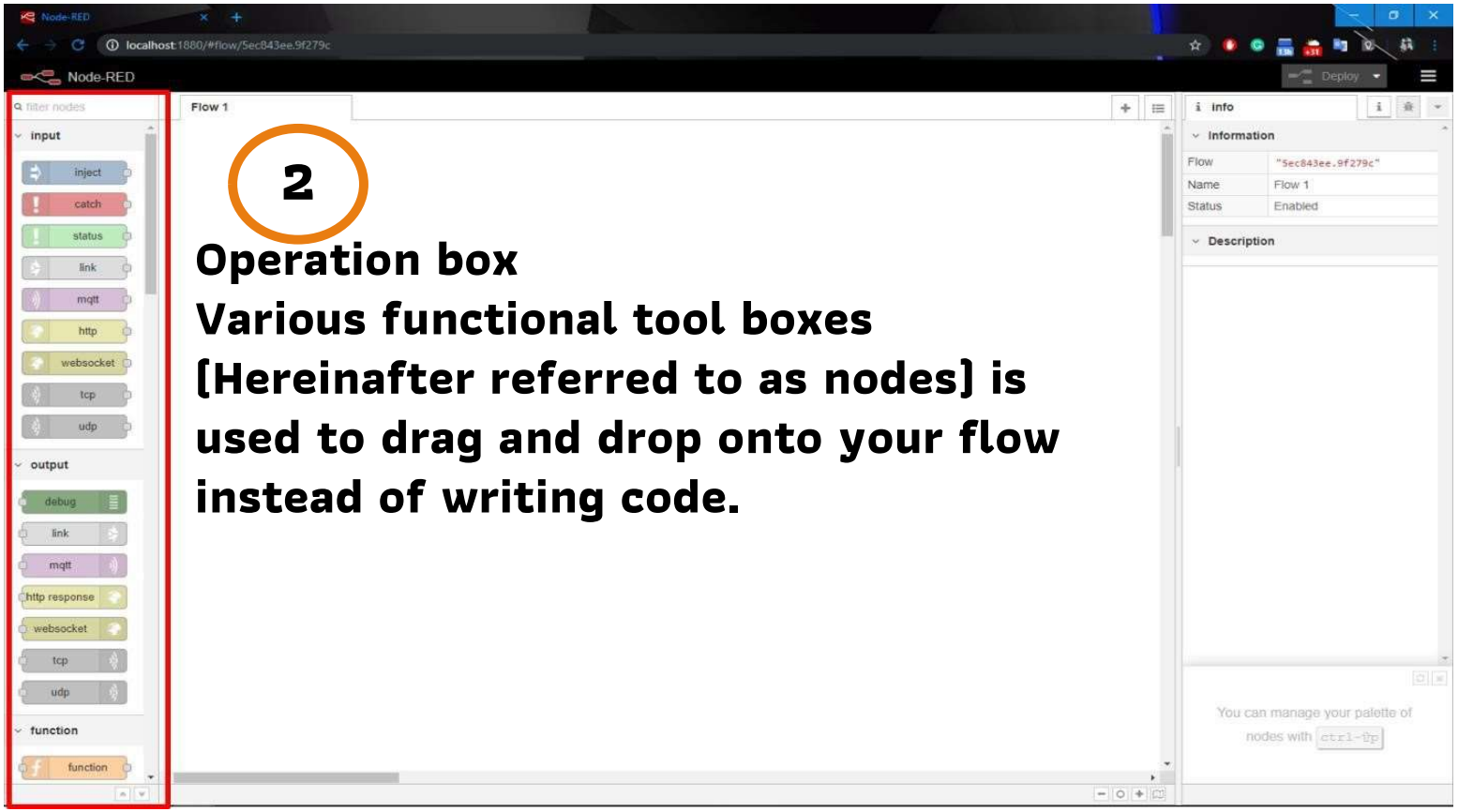
Node-RED

Component



Node-RED

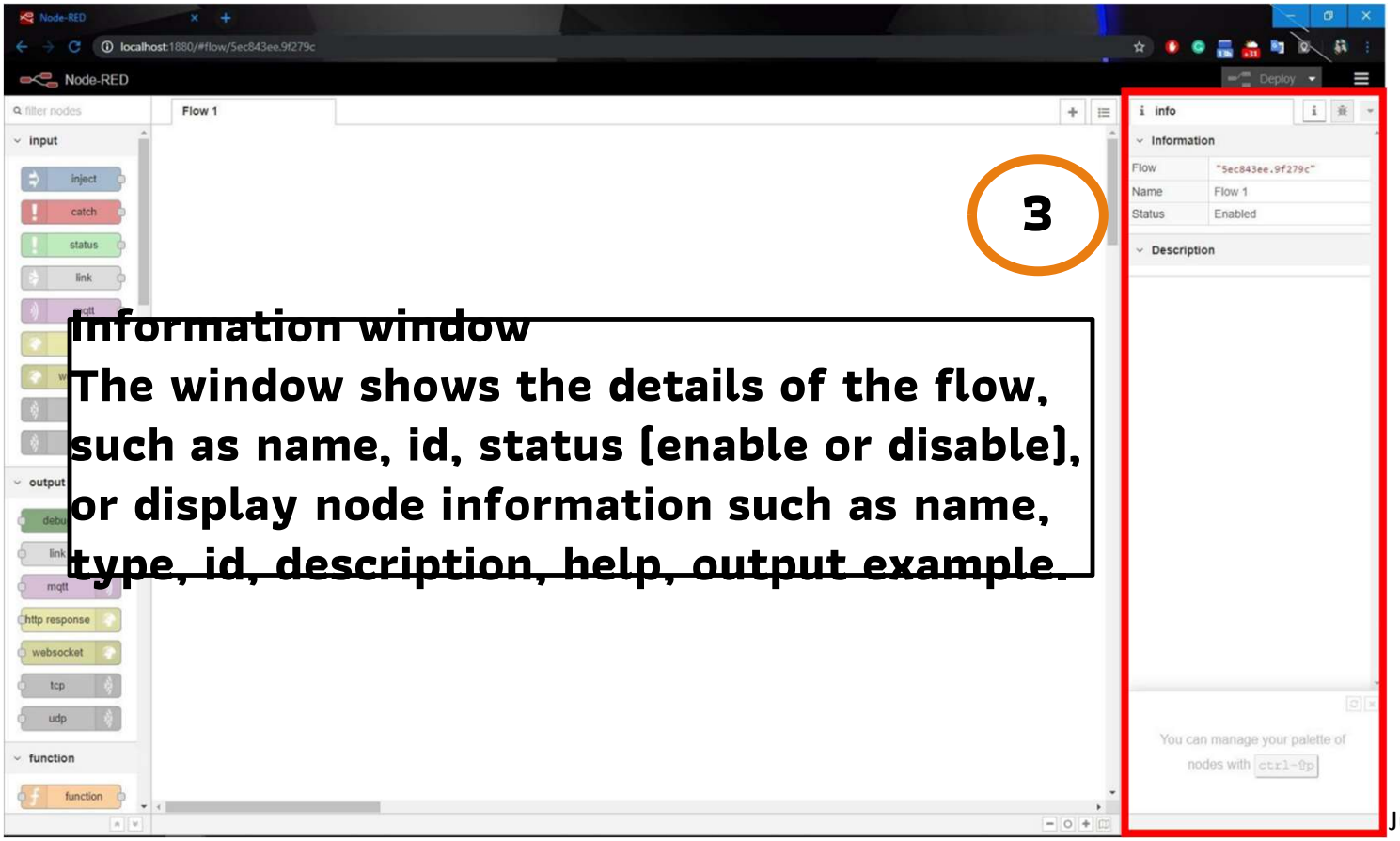
Component



Operation box
Various functional tool boxes [Hereinafter referred to as nodes] is used to drag and drop onto your flow instead of writing code.

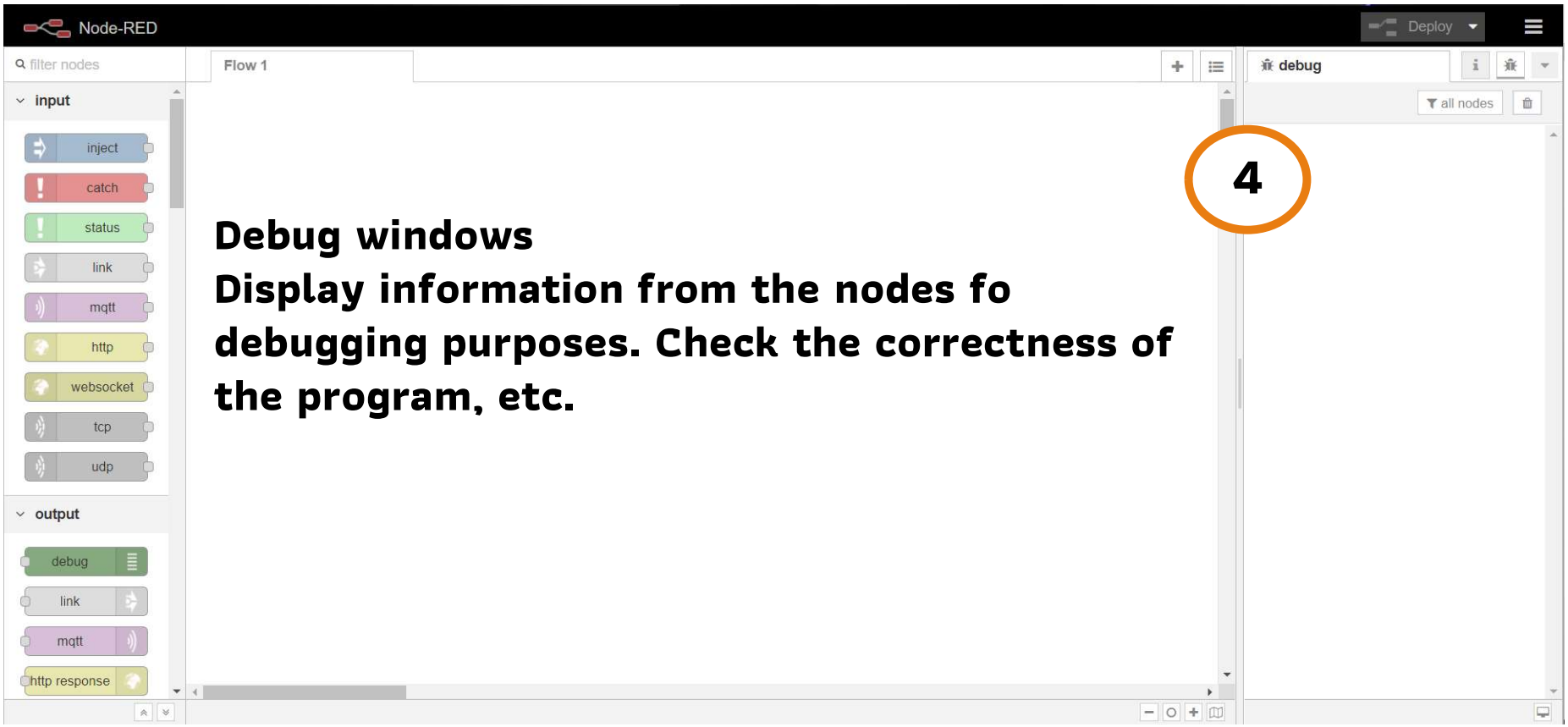
Node-RED

Component



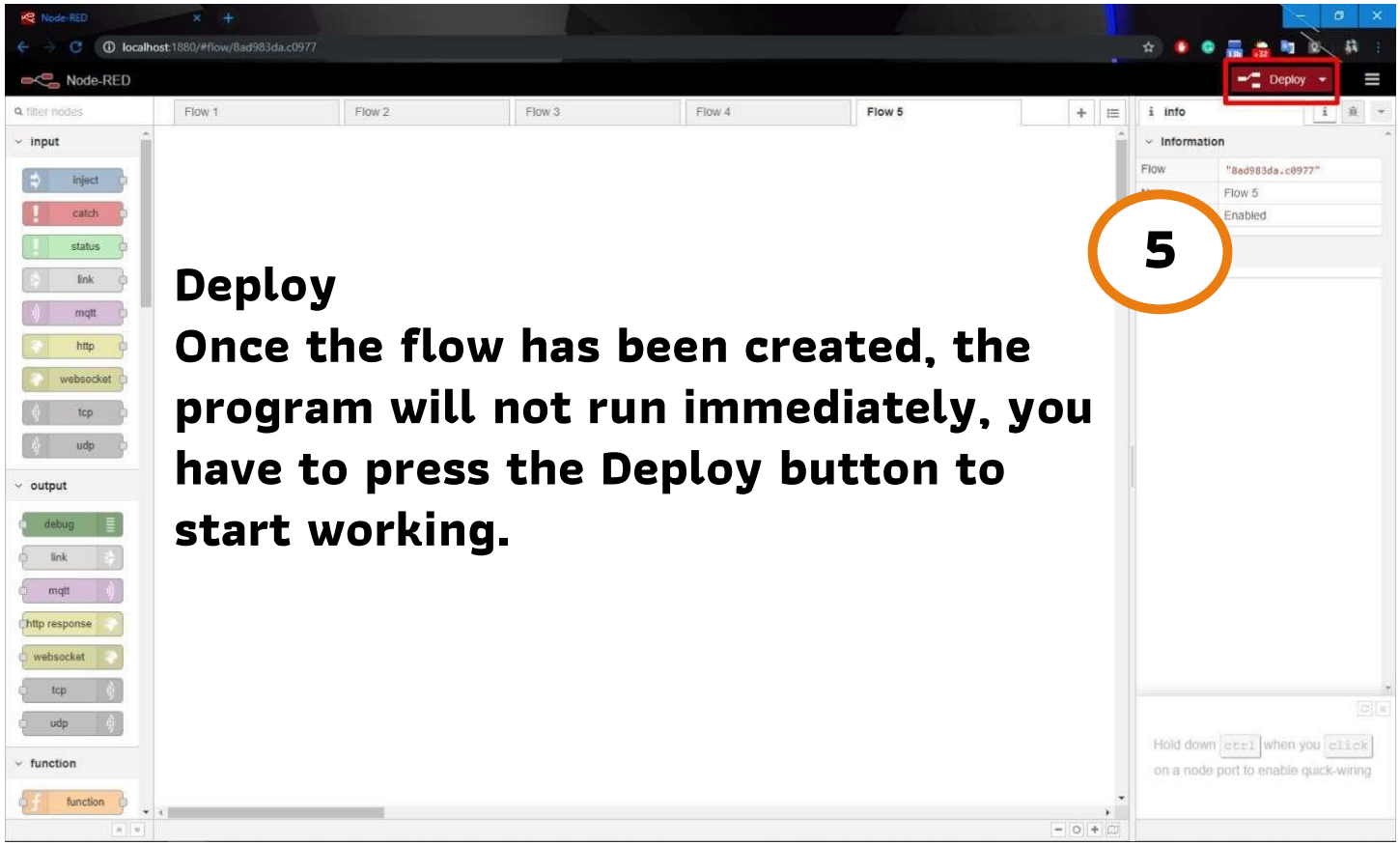
Node-RED

Component



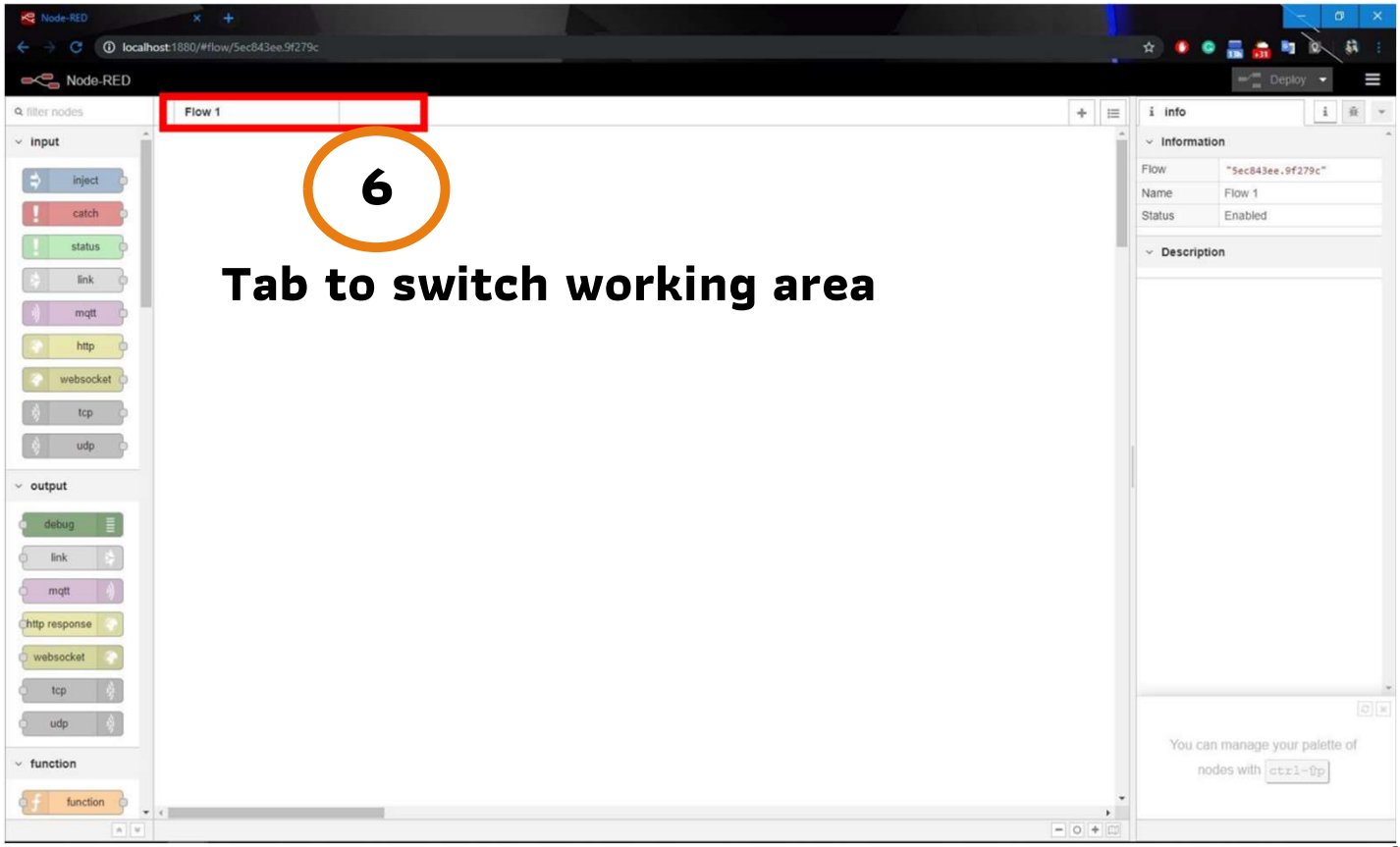
Node-RED

Component

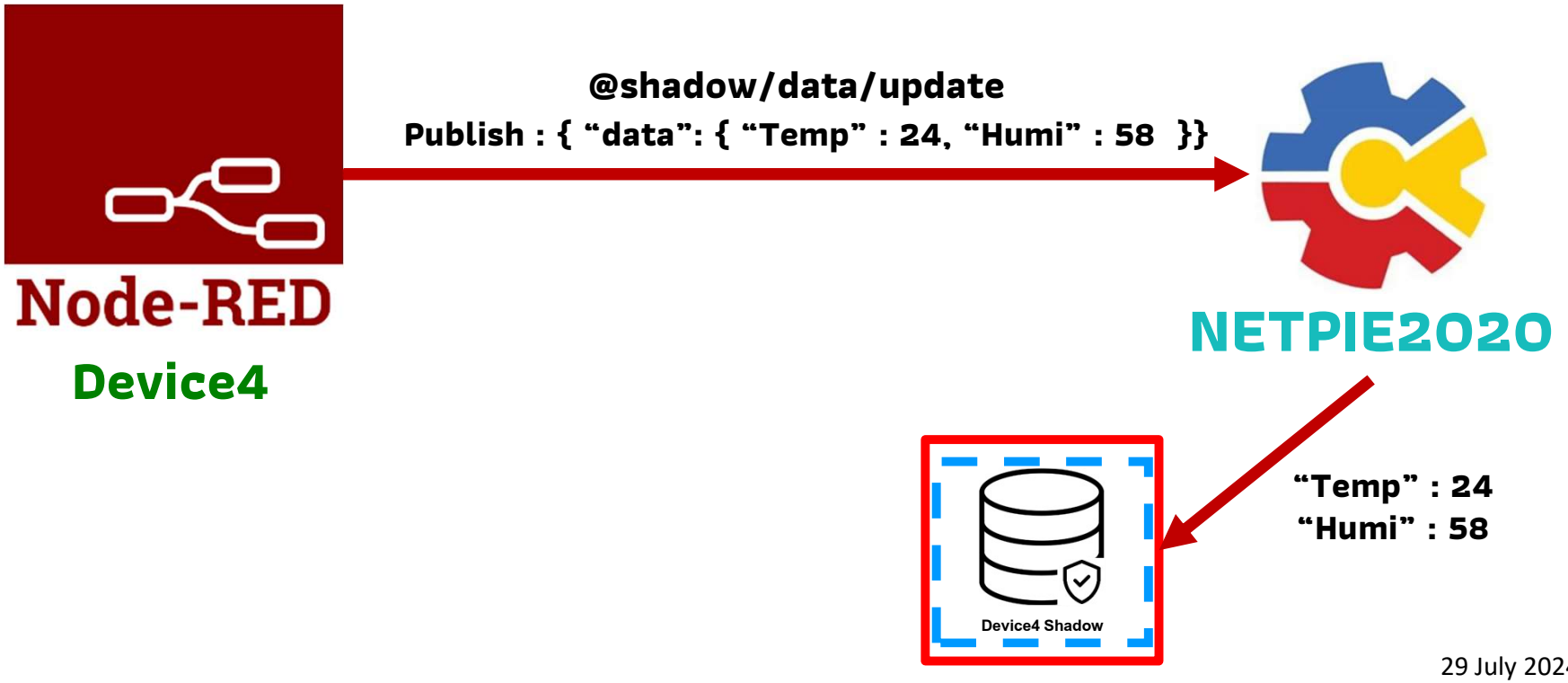


Node-RED

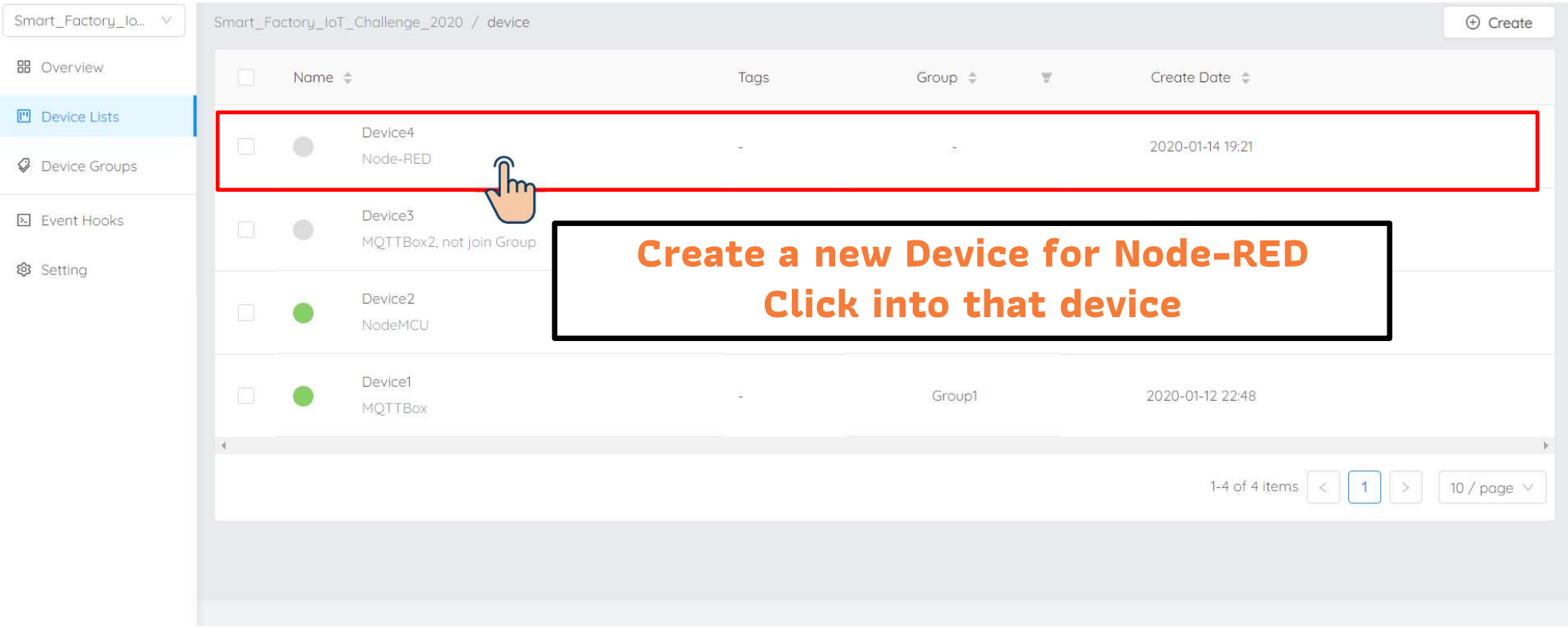
Component



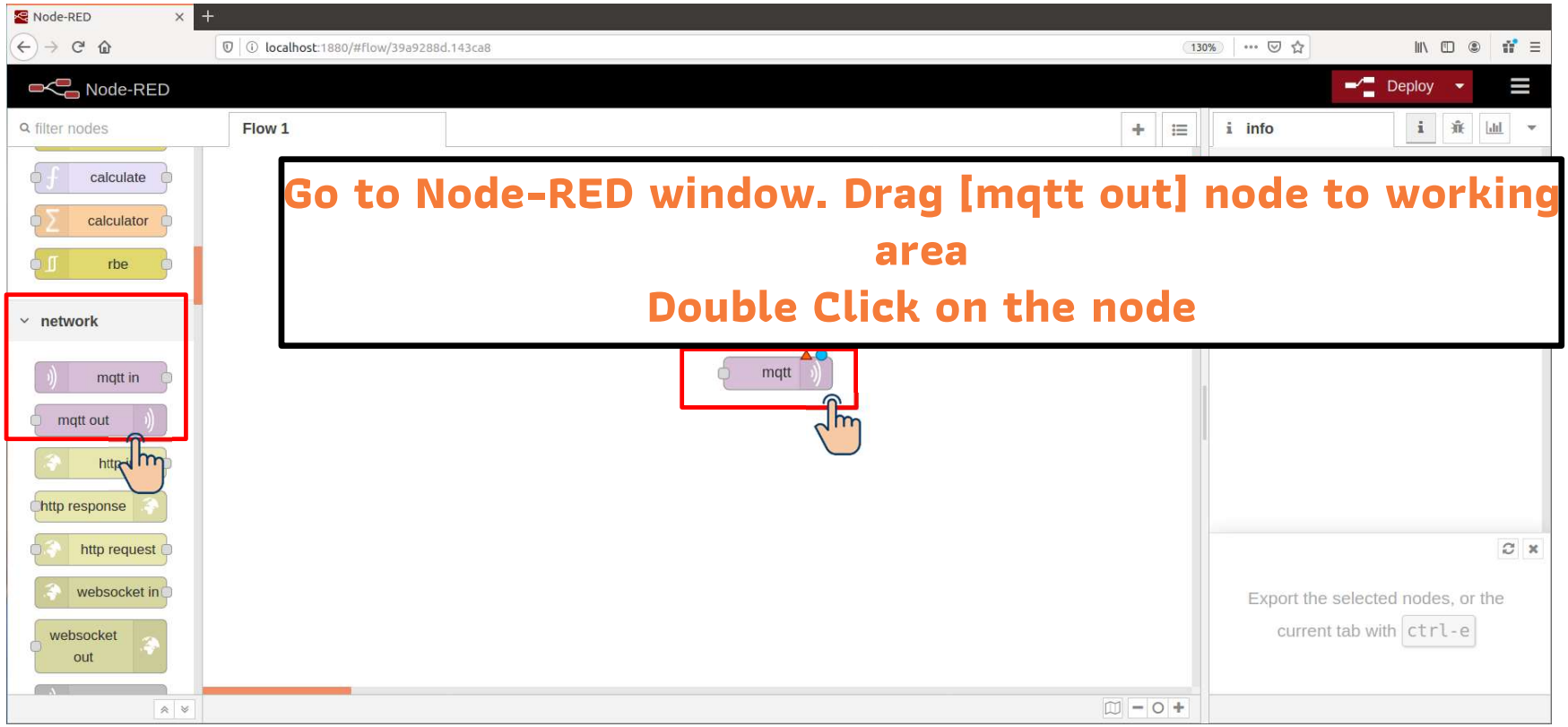
Workshop 9 : Connect NETPIE2020 with Node-RED



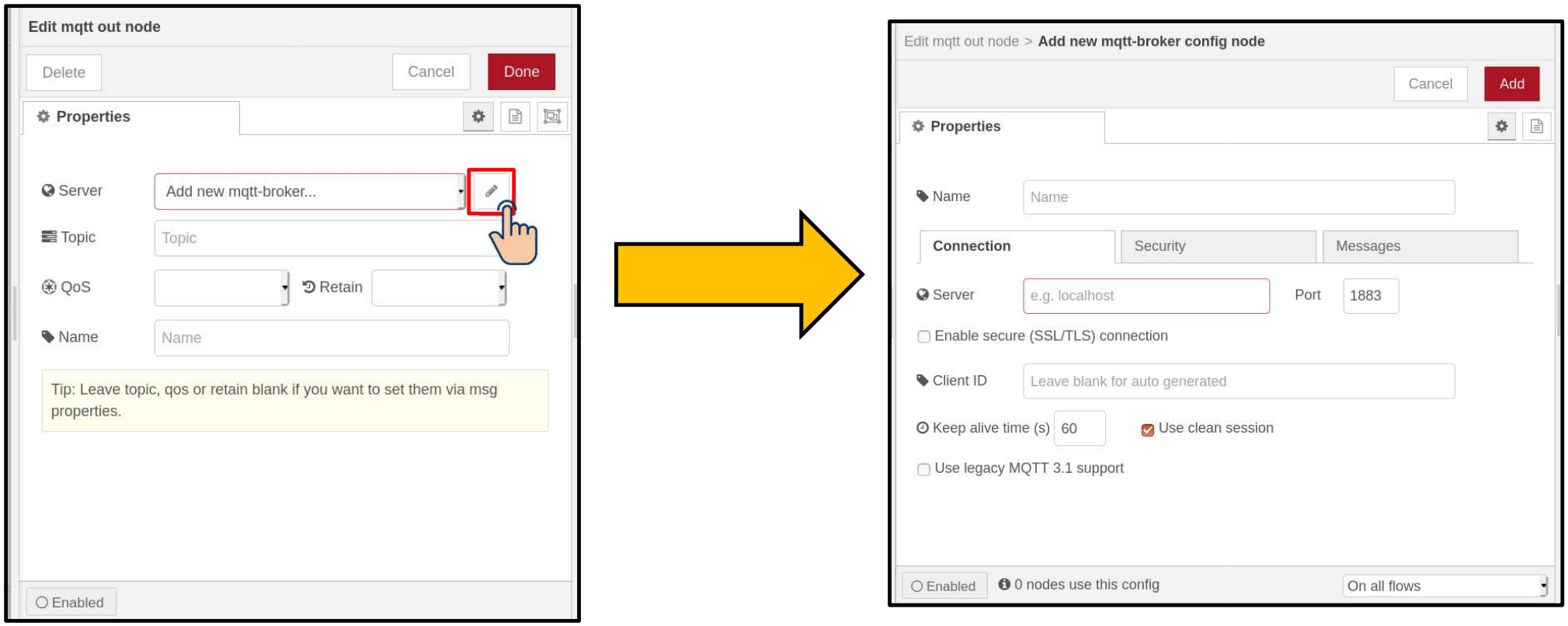
Workshop 9 : Connect NETPIE2020 with Node-RED



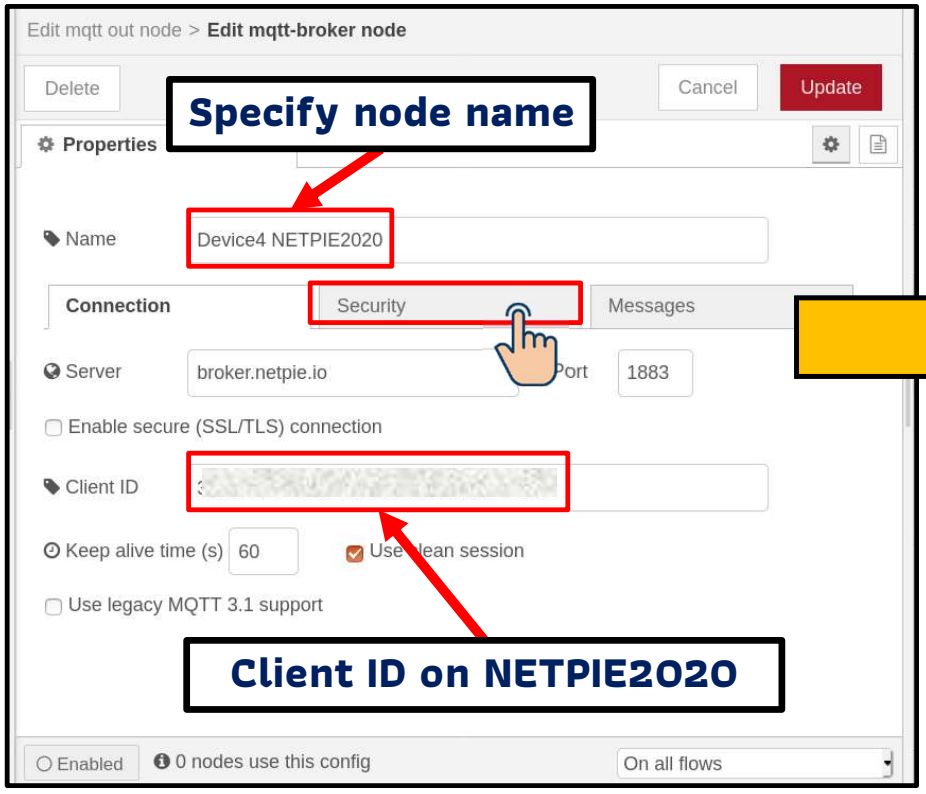
Workshop 9 : Connect NETPIE2020 with Node-RED

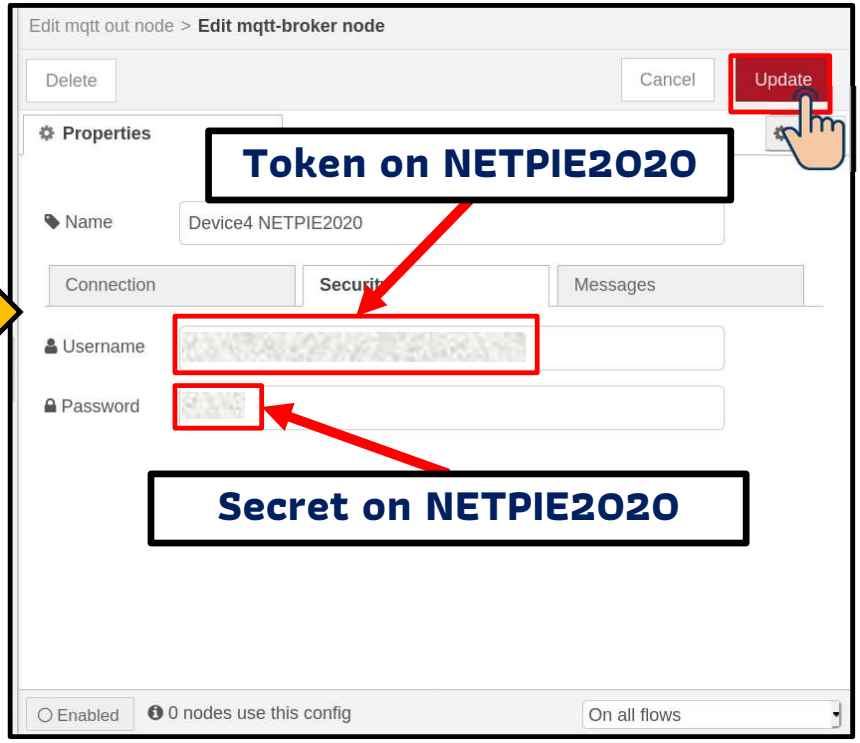


Workshop 9 : Connect NETPIE2020 with Node-RED

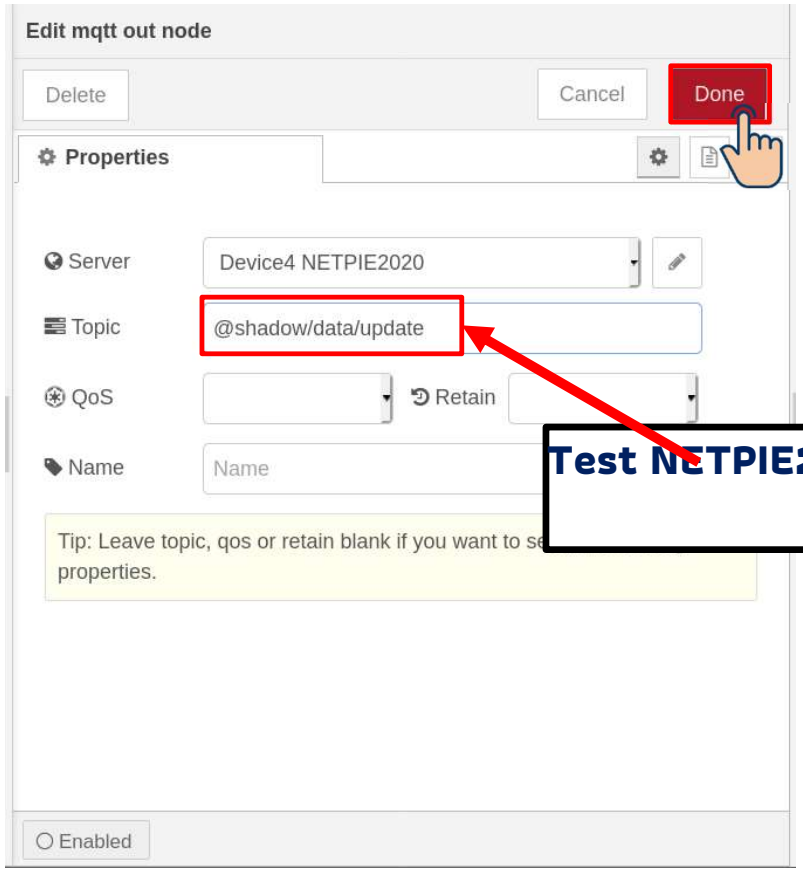


Workshop 9 : Connect NETPIE2020 with Node-RED



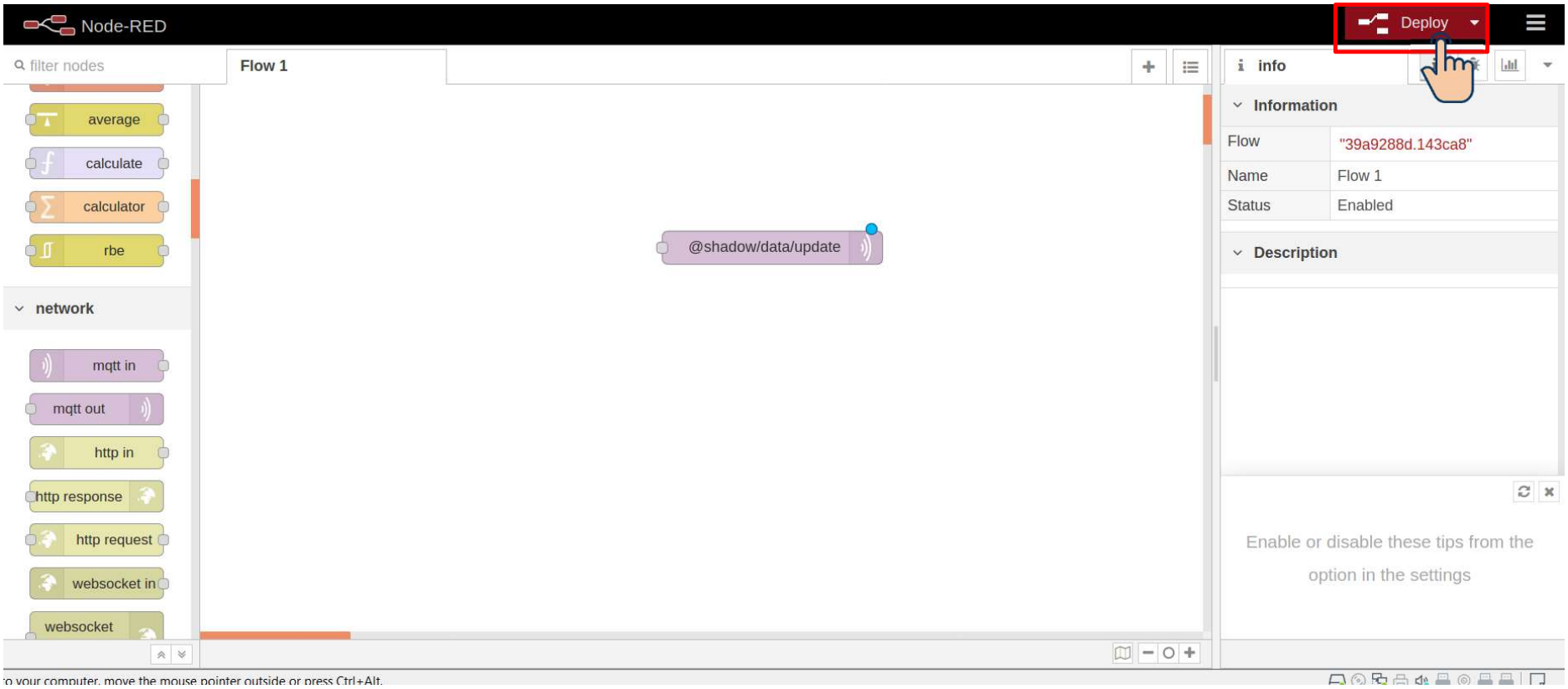


Workshop 9 : Connect NETPIE2020 with Node-RED

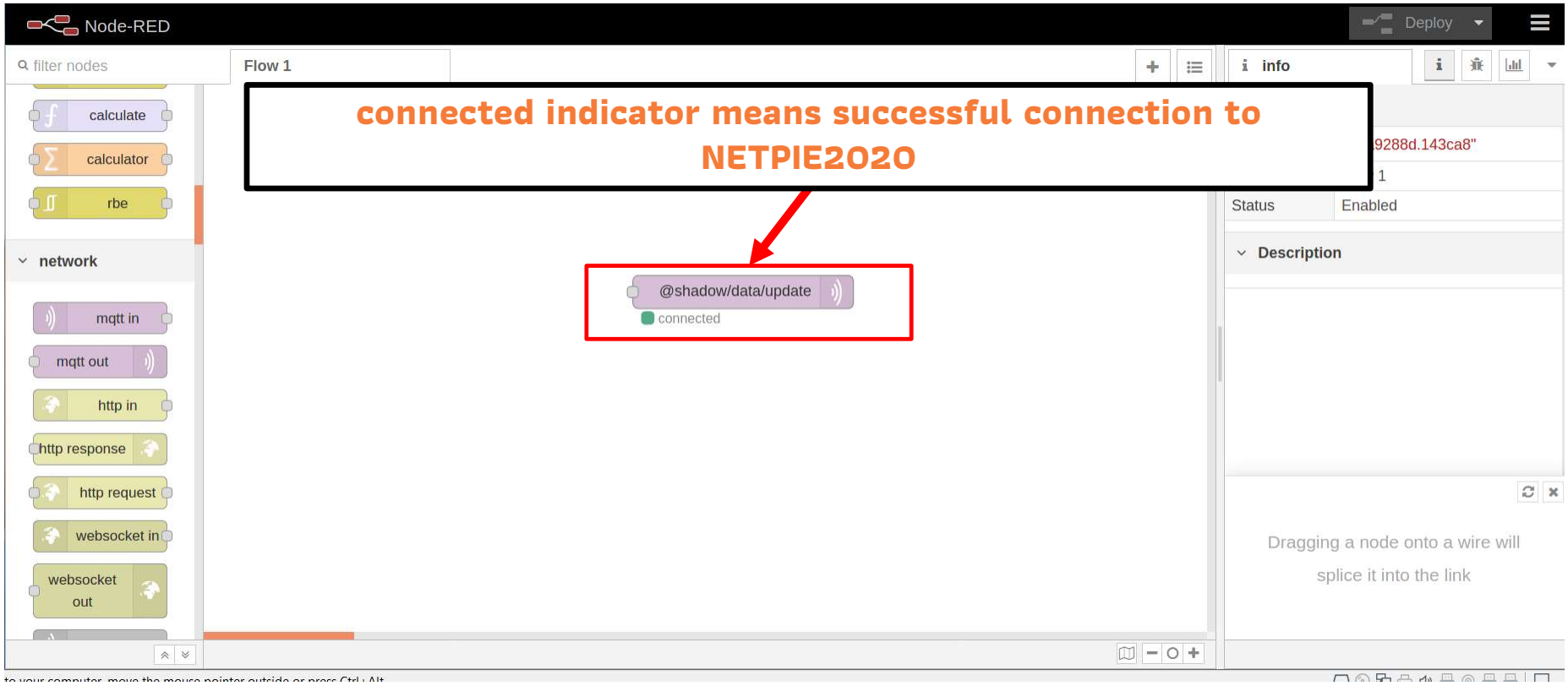


Test NETPIE2020 on Node-RED by updating data to Device Shadow.
Specify topic : @shadow/data/update

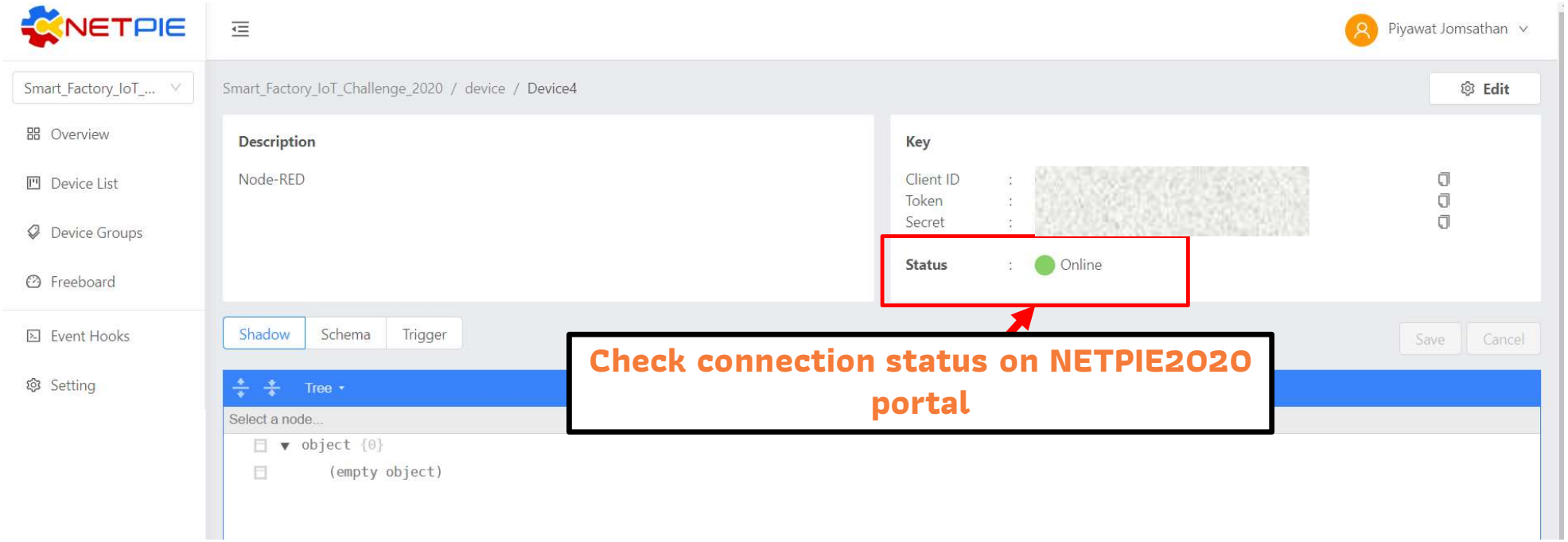
Workshop 9 : Connect NETPIE2020 with Node-RED



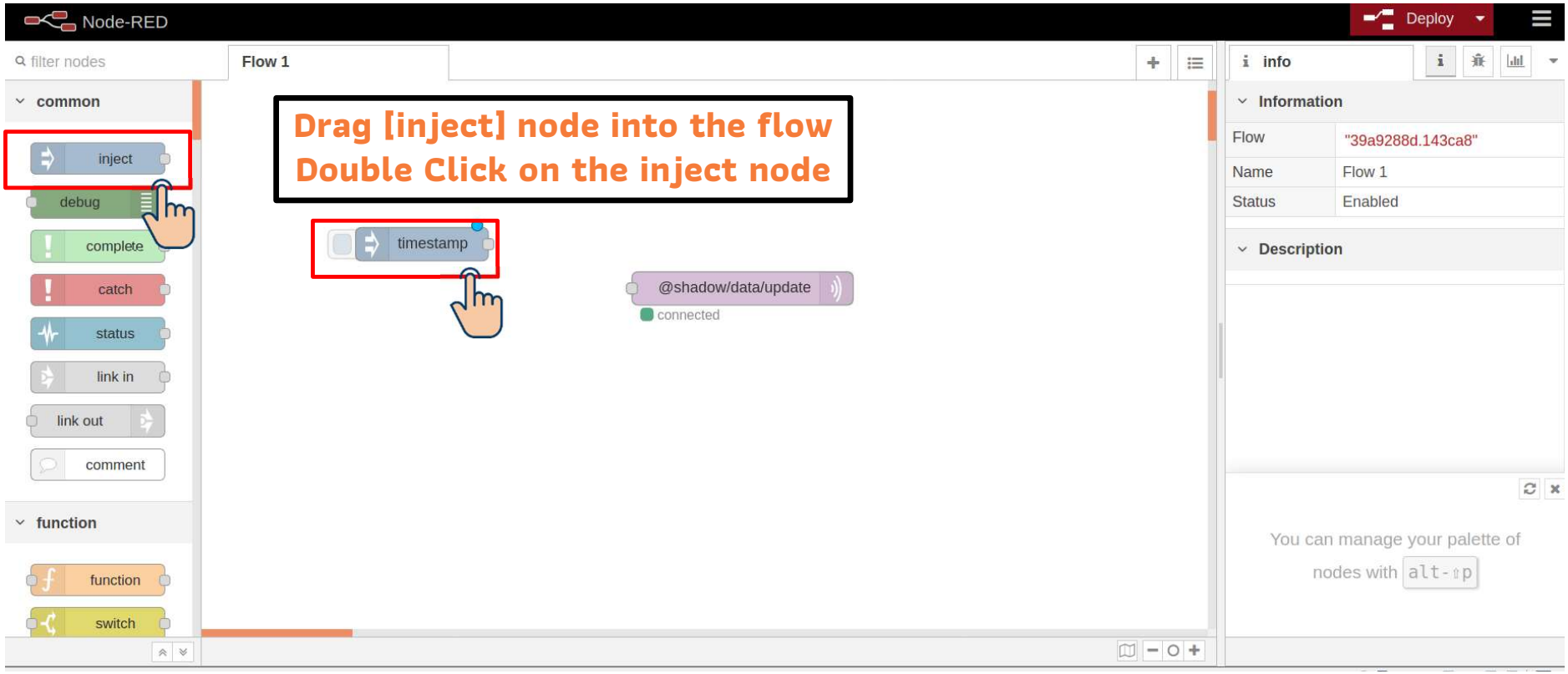
Workshop 9 : Connect NETPIE2020 with Node-RED



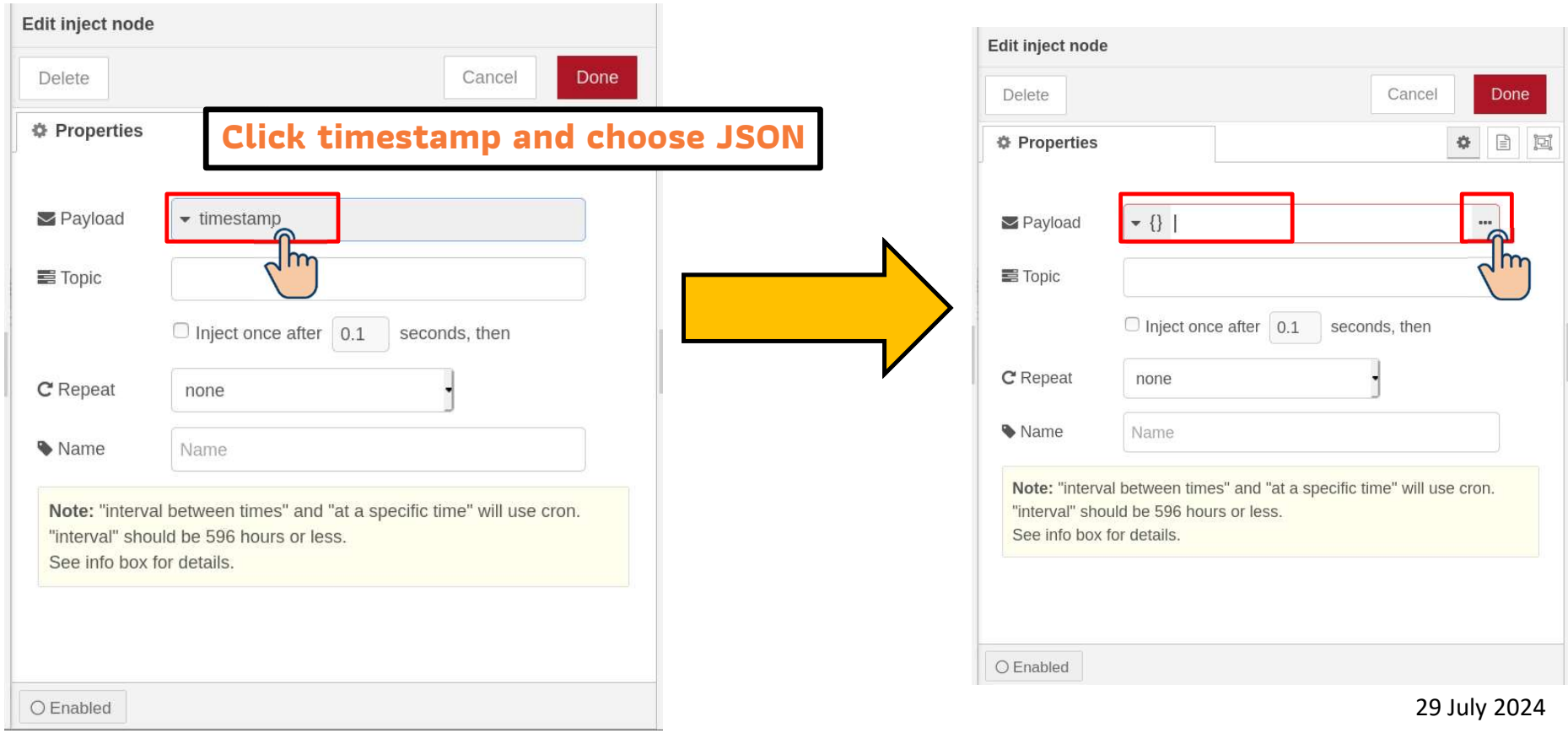
Workshop 9 : Connect NETPIE2020 with Node-RED



Workshop 9 : Connect NETPIE2020 with Node-RED



Workshop 9 : Connect NETPIE2020 with Node-RED

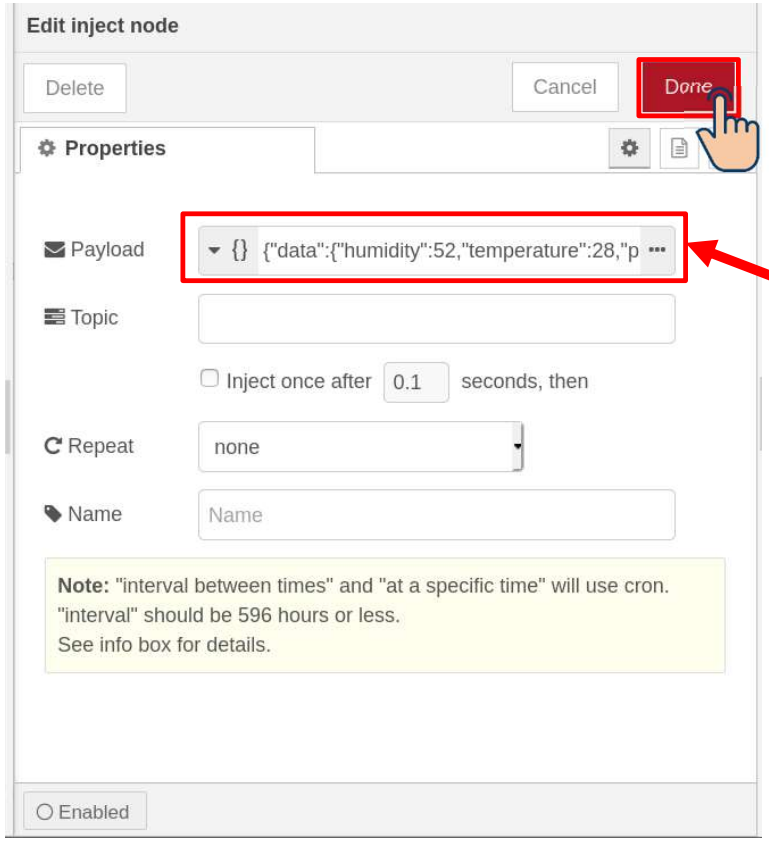


Workshop 9 : Connect NETPIE2020 with Node-RED

JSON code example
{ "data" : { "humidity" : 52, "temperature" : 28, "place" : "NECTEC Thailand" }}

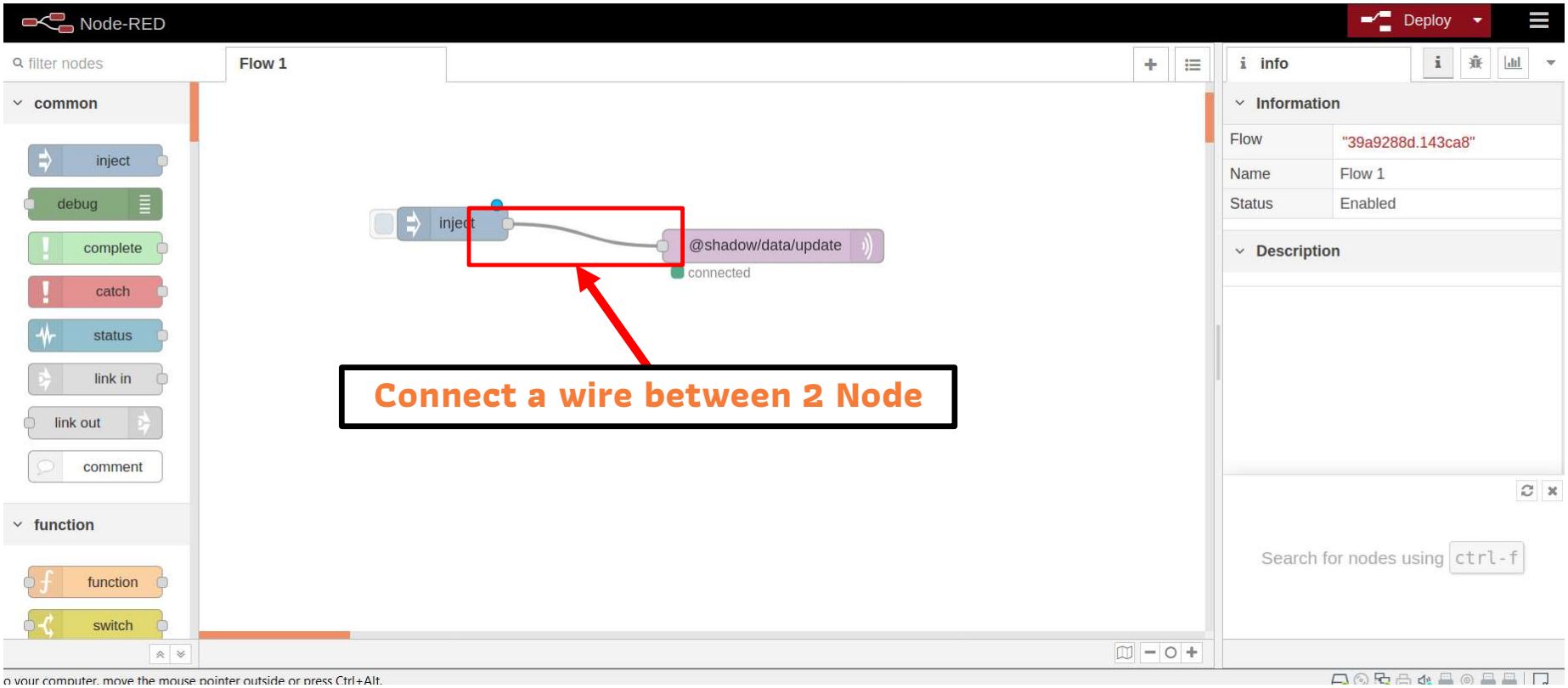
Copy JSON code example above to the editor Then click [Done]

Workshop 9 : Connect NETPIE2020 with Node-RED

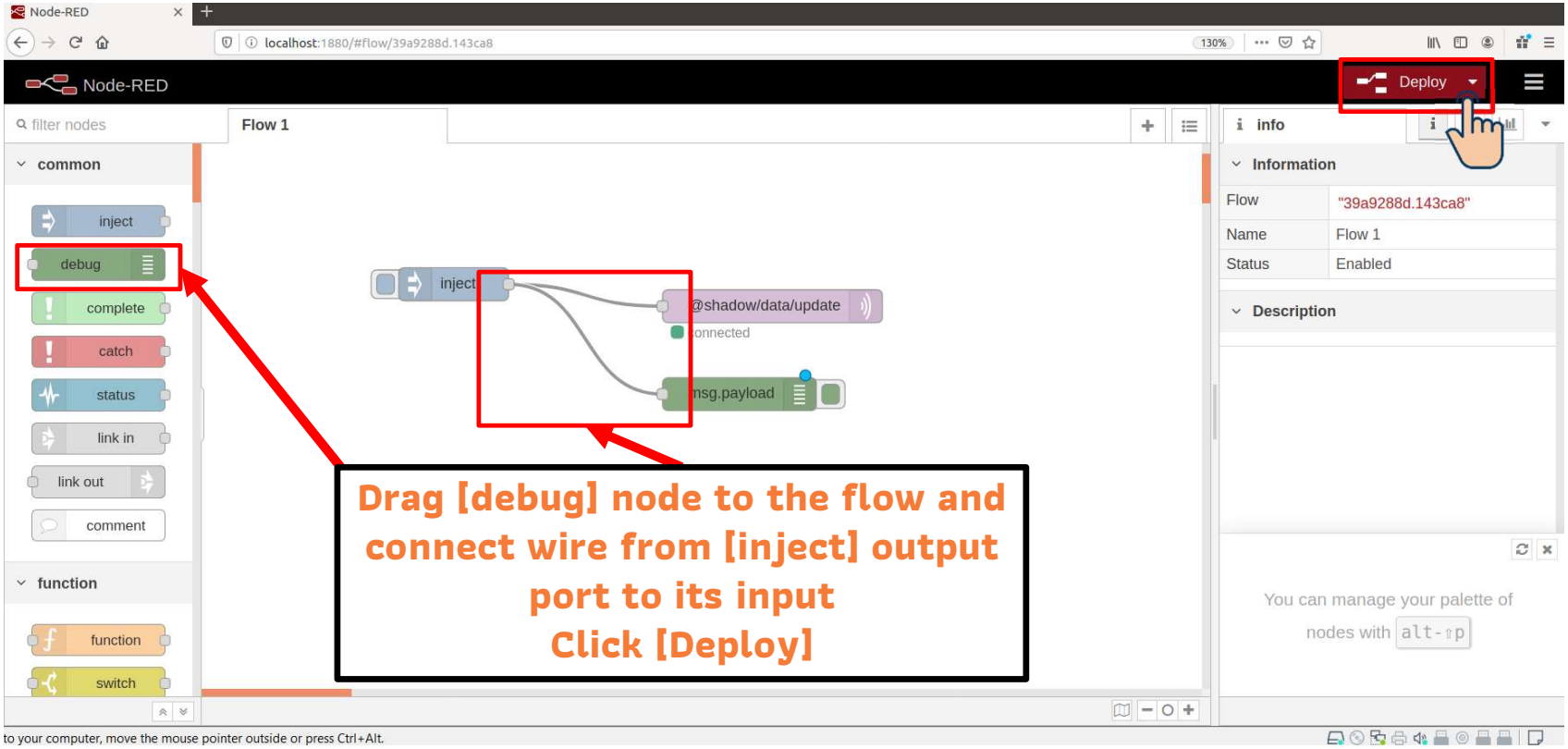


the message to be sent

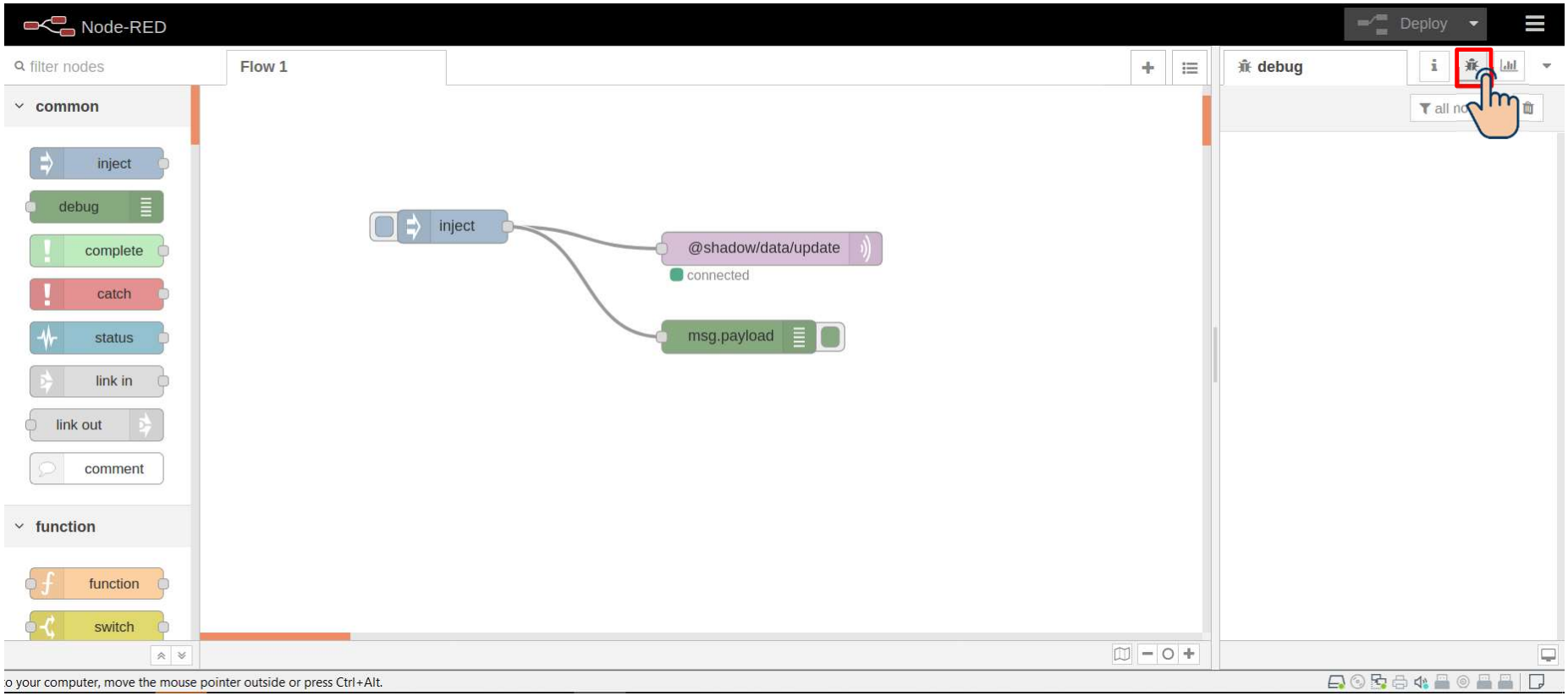
Workshop 9 : Connect NETPIE2020 with Node-RED



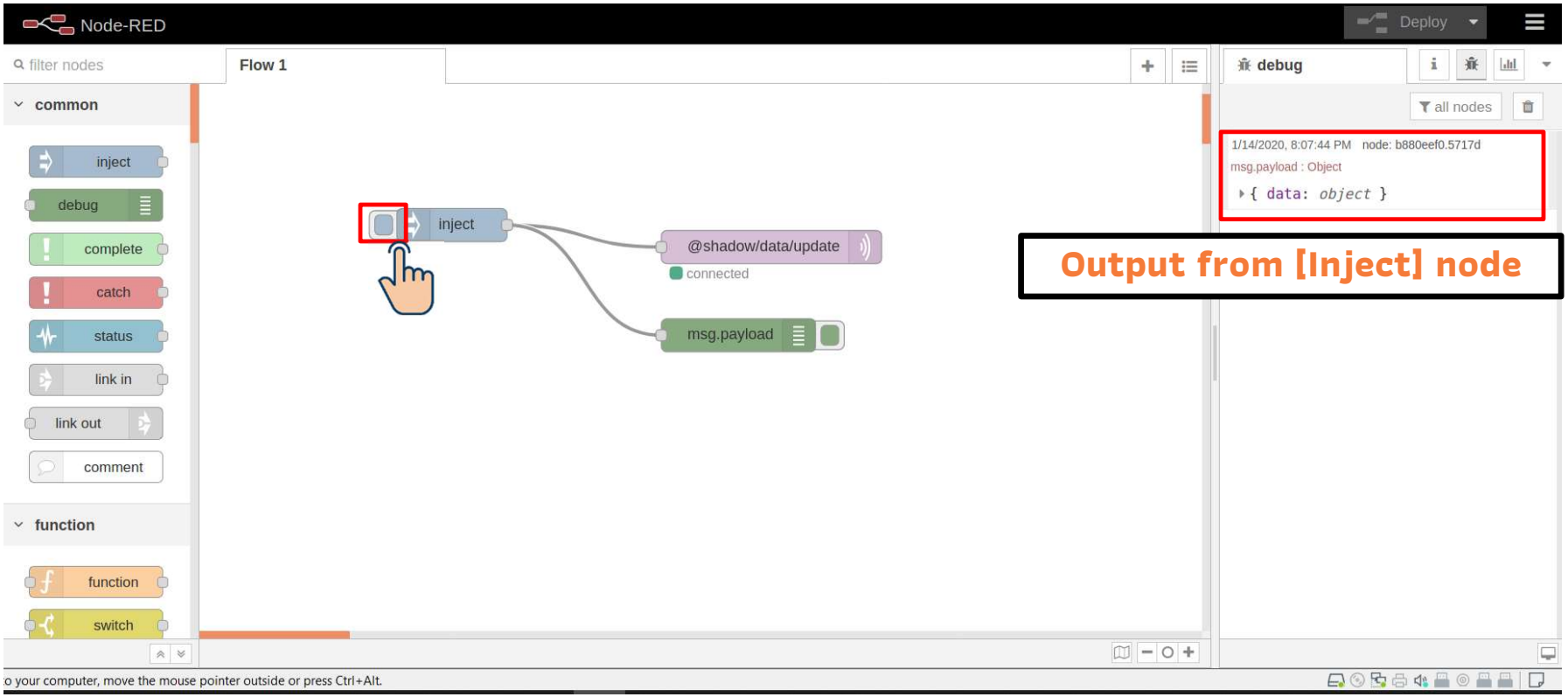
Workshop 9 : Connect NETPIE2020 with Node-RED



Workshop 9 : Connect NETPIE2020 with Node-RED



Workshop 9 : Connect NETPIE2020 with Node-RED



Workshop 9 : Connect NETPIE2020 with Node-RED

Can follow
Tutorials and Example Projects from

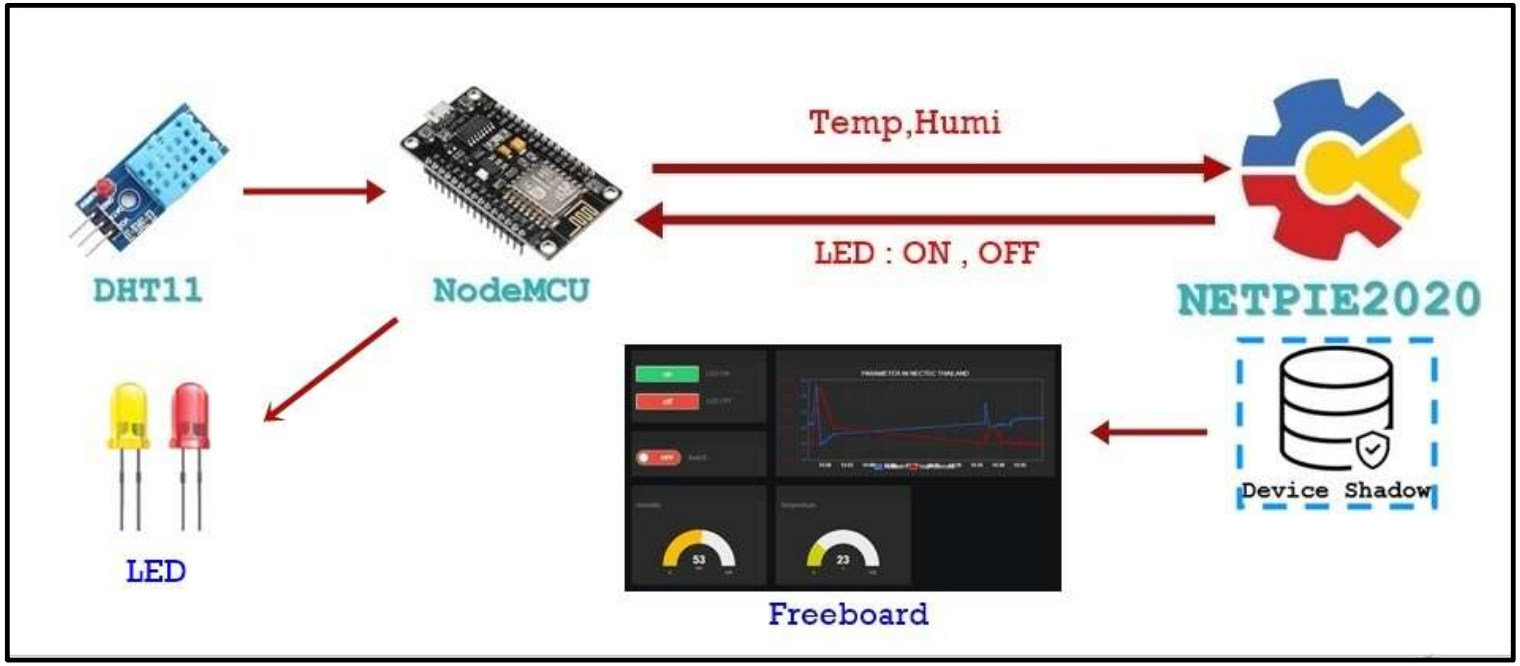
https://netpie.io/tutorials

https://netpie.io/exampleprojects

Coming Soon...

Example of NETPIE 2020 connection with various hardware platforms

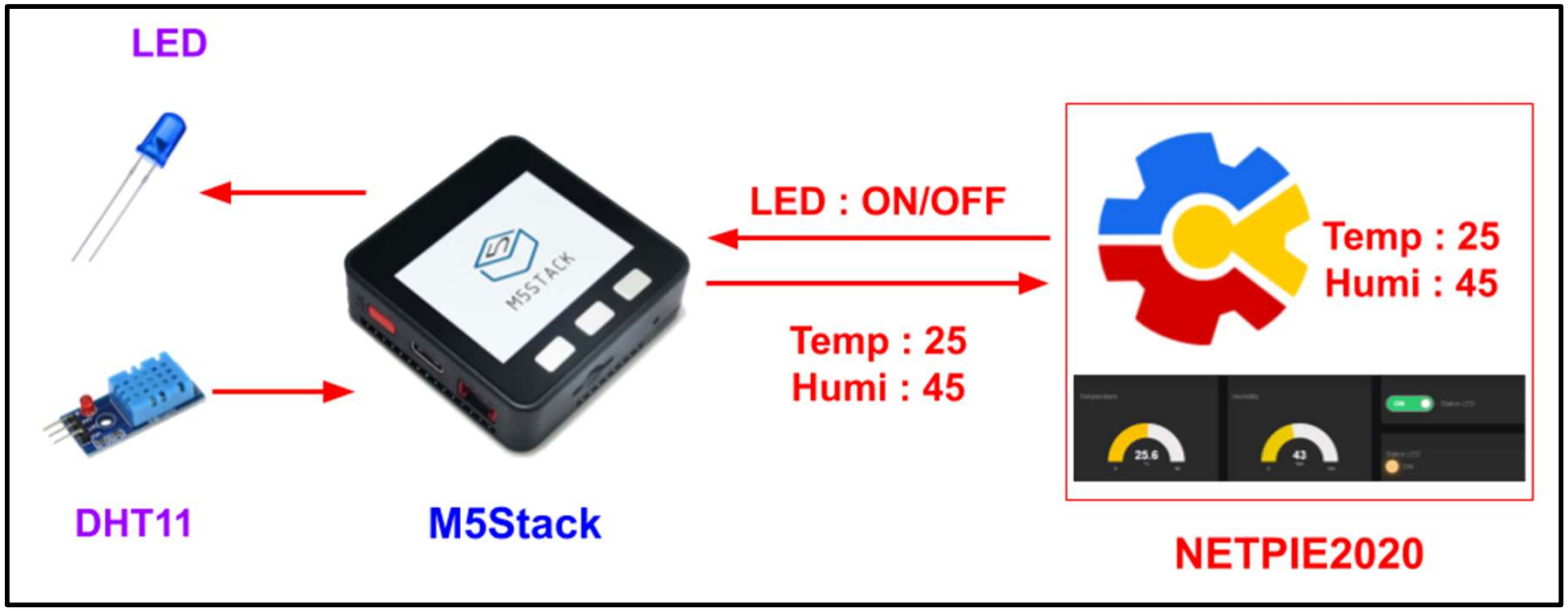
Hardware : NodeMCU(ESP8266)



Coming Soon...

Example of NETPIE 2020 connection with various hardware platforms

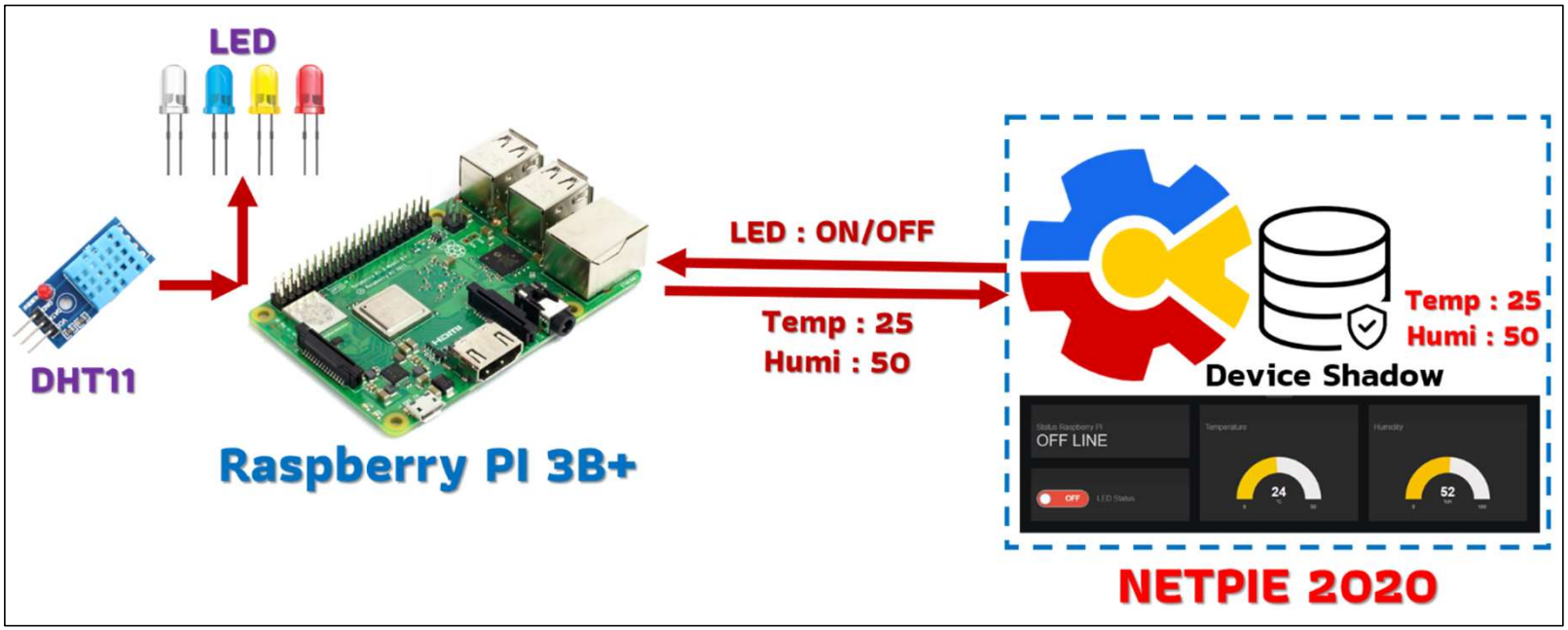
Hardware : M5Stack(ESP32)



Coming Soon...

Example of NETPIE 2020 connection with various hardware platforms

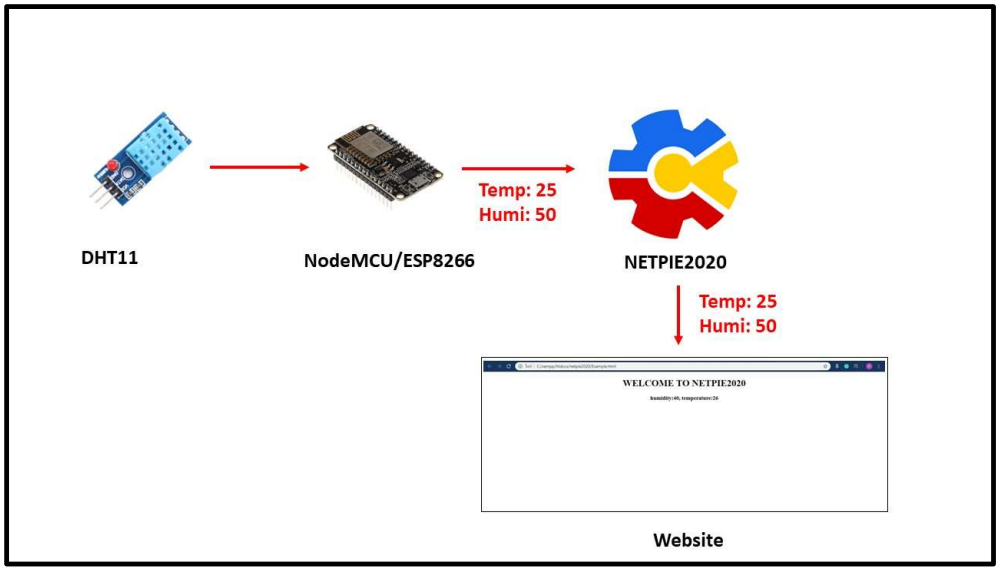
Hardware : Raspberry PI



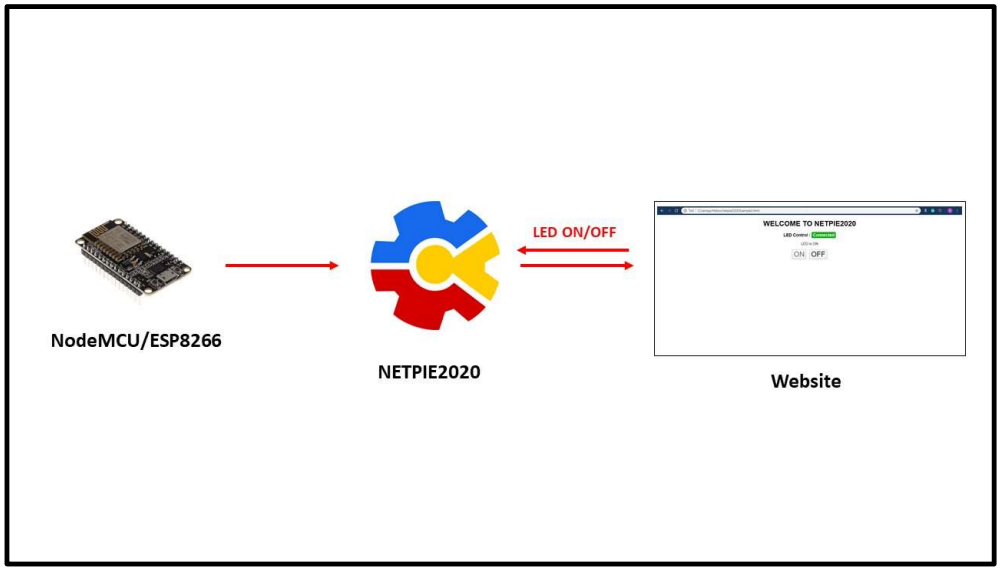
Coming Soon…

Example of NETPIE 2020 connection with various hardware platforms

Monitor sensor values on Web Browser



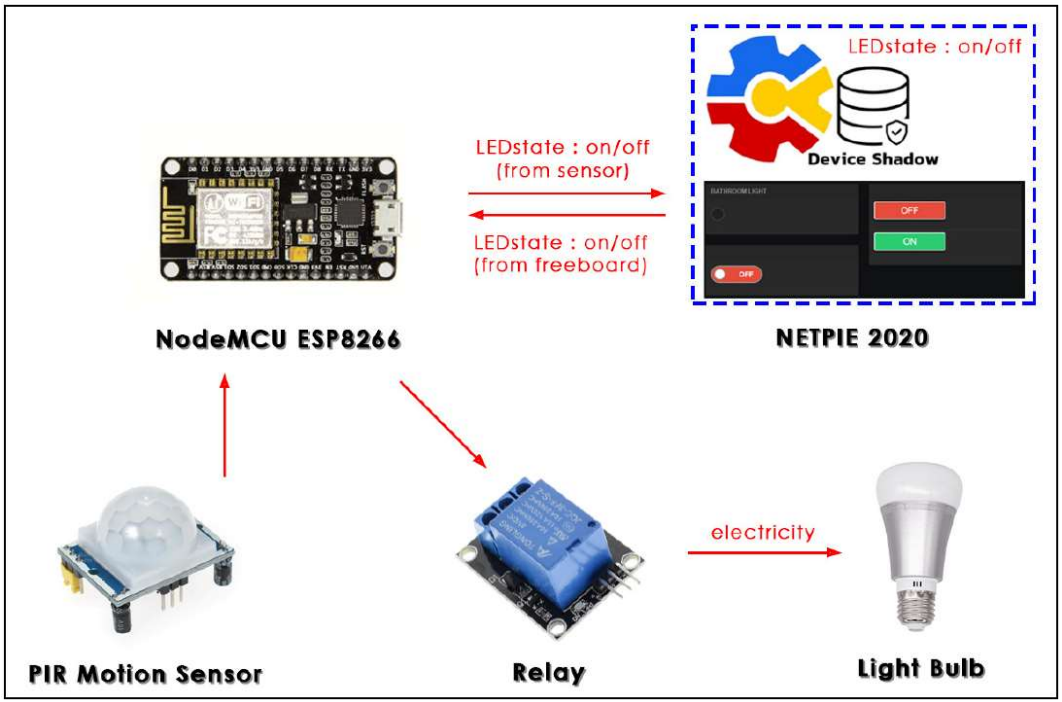
Control devices via Web Browser



Coming Soon...

Example of NETPIE 2020 connection with various hardware platforms

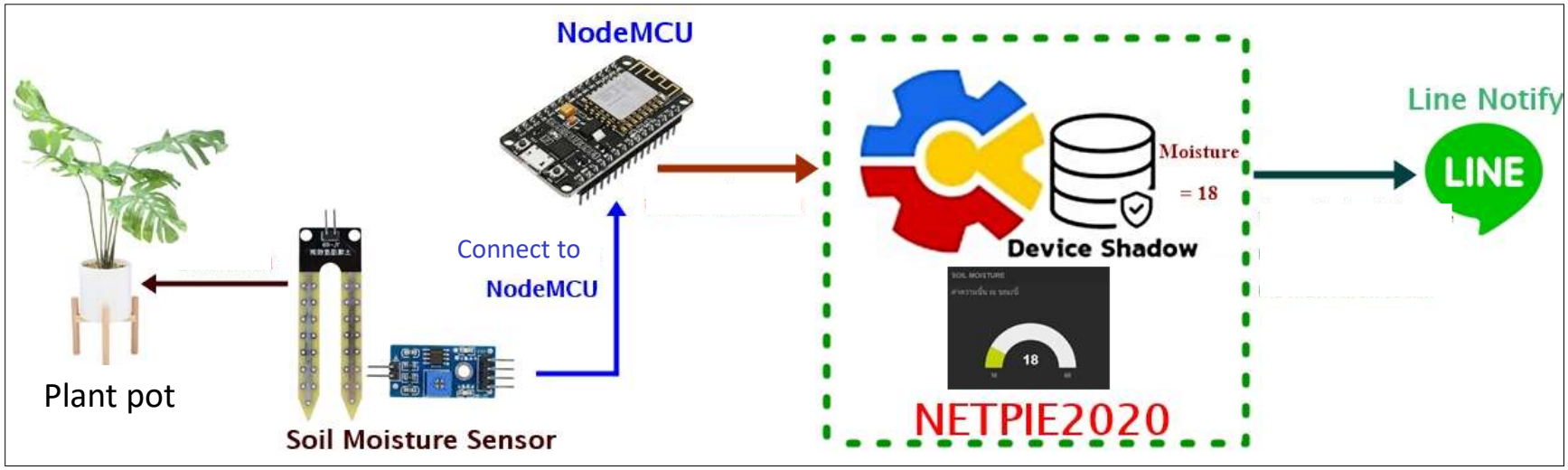
Smart Home



Coming Soon...

Project examples that use NETPIE2020

Smart Farm



Coming Soon...


NETPIE
2020
From Makers Nation
Toward Smart Nation
New Version
Grand opening: 20.02.2020

Translation to English by Dr.Varodom Toochinda
Dept. of Mechanical Engineering,
Kasetsart University

Thank You