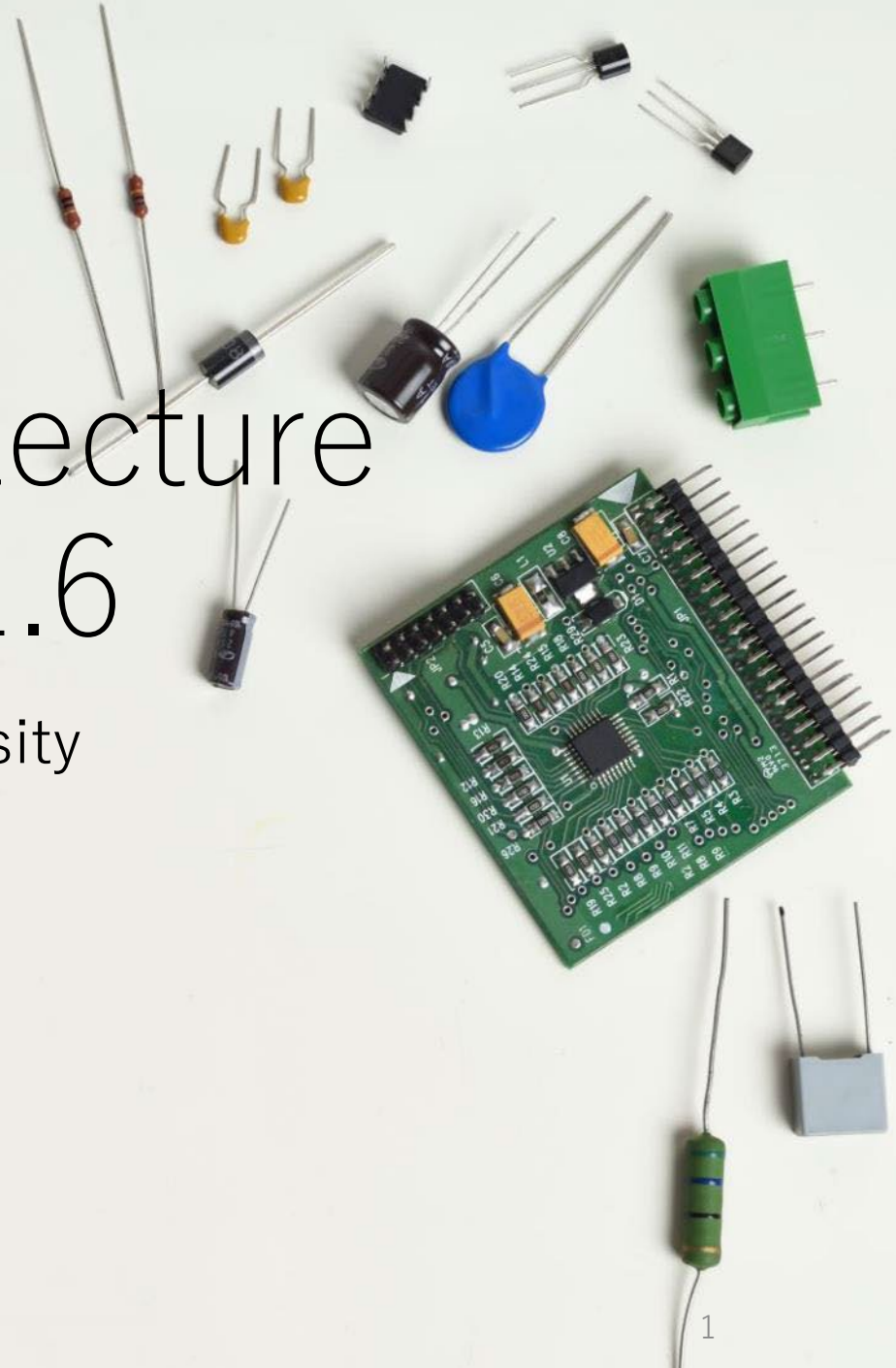
1451 Hands-on Lecture
IEEE P1451.1.6

Hiroaki Nishi, Keio University



Directions

- The program will be facilitated in a hands-on style.
- In two-day lectures, we will design your own NCAP application and connect with sensors and actuators over the Internet.
- We will use Low-Code style design (Node-RED).
  - It is needless to be familiar with programming language.
  - It is needless to know about the Internet protocol.
- However, we will develop:
  - Installing OS and set-up real server
  - Set-up all design environments and applications by yourself
  - Design your own 1451.1.6 network application
- We have a deadline
  - Please do not hesitate asking if you have a trouble.
  - Due to the progress of the program, delays may result in joining a breakout session with different content.

Preparation in advance

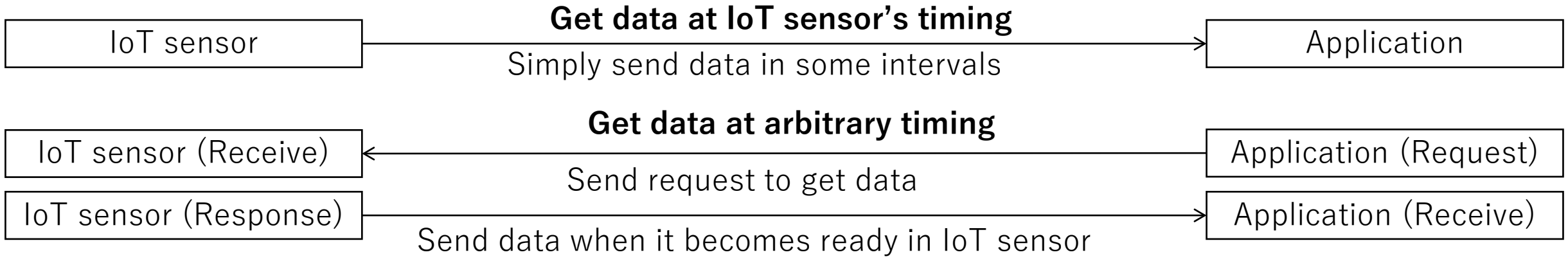
- Prepare one Raspberry-Pi per person. You will also need a USB keyboard, mouse, HDMI monitor, and HDMI cable. If you are using your own PC, use that.
  - https://www.raspberrypi.com/documentation/computers/getting-started.html
  - It does not matter if you are at the university or at home, you should participate with a Raspberry-Pi with internet access available.
- If you do not use a Raspberry-Pi, you may install the software using the free Oracle Cloud. It is free, but a credit card registration is required.
- Either wired or WiFi Internet connection is acceptable. WiFi is preferred for continuity of content, but is not recommended as it is irrelevant to the content of the program.
- Please make sure that you have installed the Raspberry-Pi OS on your Raspberry-Pi and that you can connect to the Internet according to the materials distributed in advance.
- It is possible to design hands-on without a connection, but it is not possible to check the operation or test the connection with other students.

What will we do?

- Learn about 1451.1.6
- Learn about MQTT
- Install Node-RED to Raspberry-Pi
- Learn about Node-RED
- Install required libraries to Node-RED
- Access to sensor (using direct method) and design visualization application
- Access to sensor (using D0 method)
- Exchange message (sensor data) with others

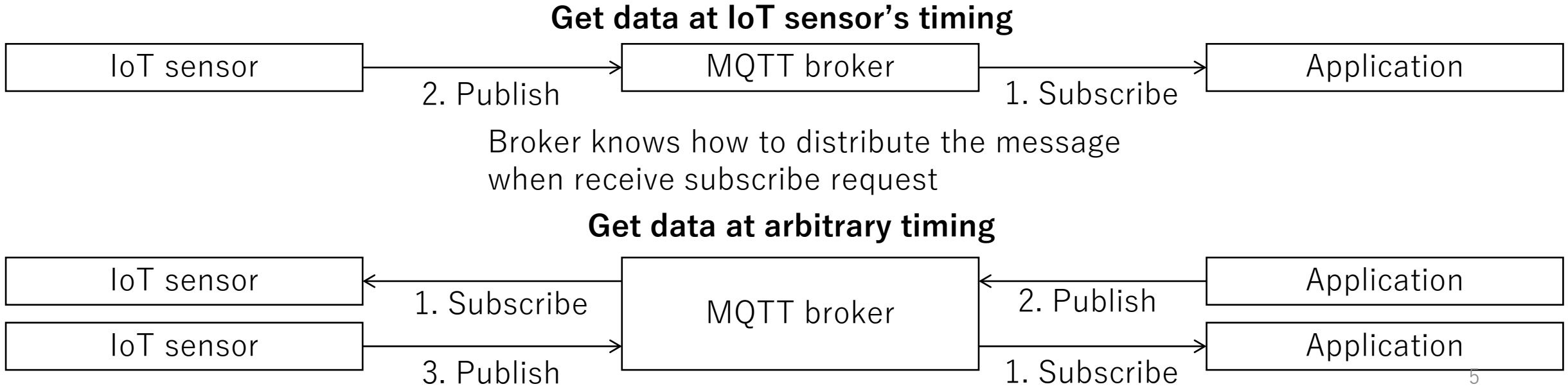
What is MQTT (General communication model)

- General communication model (request-response communication model)



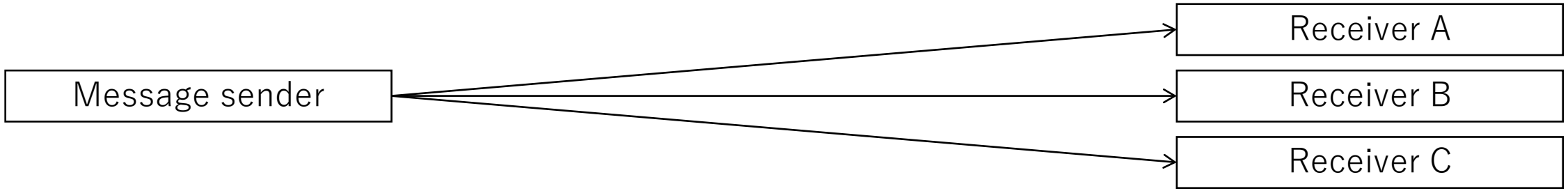
Q: Which one can reduce the power consumption of IoT sensor?

- Publish-subscribe communication model



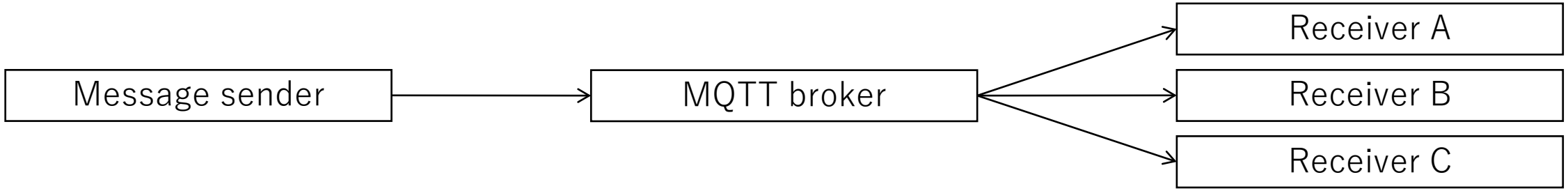
What is MQTT (Muticast / Broadcast)

- General communication model



Single casts must be repeated to convey information to multiple receiving nodes

- Publish-subscribe communication model



Only one message is enough to send all subscribers

Q: Which one can reduce the power consumption of IoT sensor?

What is the merit of MQTT (general perspectives)

- Most IoT systems support MQTT because…
  - Simple protocol design: MQTT header size is as small as 2 bytes minimum.
  - MQTT uses a broker server to distribute the message from an MQTT client.
    - When multicasting or broadcasting messages, it is required to manage a protocol to know all IP addresses for the network nodes and generally to send the message one by one for all the network nodes.
  - Low-power design
    - In general communication models, the power cost for processing and network communication to distribute messages increases linearly according to the number of network nodes.
    - MQTT broker only receives one publish message, and the broker distributes the message to all subscribers.
    - In this delivery, the topic name plays an important role. MQTT message has the topic name, and the topic name is the keyword that indicates a group of nodes receiving the message. Therefore, the message distribution is completely controlled only by using the topic name.
    - The naming rules of this topic name are the essential contributions as IEEE P21451.1.6.
- IEEE P1451.1.6 standard defines the topic name rules for emulating IEEE 1451.0 message transactions to achieve IEEE 1451 functions and services.

What is the merit of MQTT (1451 perspectives)

- 1451.0 requires broadcast message to announce and discover NCAPs.
  - However, the broadcast messages are not available on the Internet because it causes DDOS attacks. All broadcast address was closed.
  - Therefore, some 1451.0 messages are not available on the Internet.
  - It is required to predefine the NCAP address as firmwares or to find NCAPs by using another protocols such as DNS.
- MQTT provides broadcast/multicast accesses in the Internet.
  - MQTT provides full functions of 1451 over the Internet.

- In this lecture we do not design announce and discover messages.
  - Refer 1451.0 document to implement these messages.

MQTT v5 is now available!

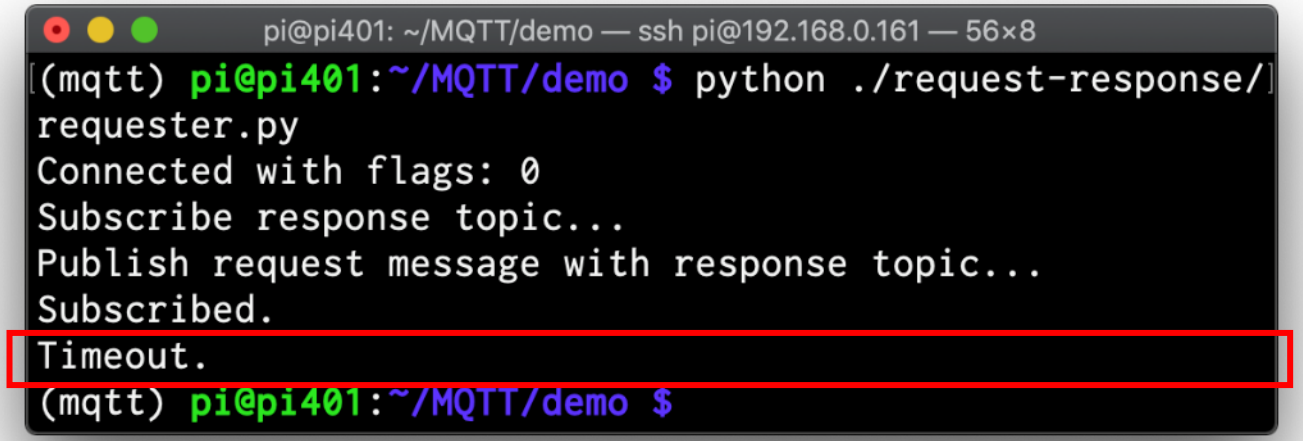
- MQTTv5 is the advanced new standard of MQTT
- The most important improvement for IEEE P1451.1.6 is the message properties of MQTT v5.
- Properties provide a new field for the message body, and the field is independently existing with the original message body. The properties give flexibilities for MQTT message chains and total protocol design.
- IEEE P1451.1.6 is targeting both uses of MQTTv3 and MQTTv5. MQTTv3 is the traditional MQTT and is used as the IoT de facto standard messaging protocol in the world.

- MQTT broker mosquitto > version 1.6 supports MQTT v5
- It enables request-response style MQTT message by using special properties.
- It also advanced in security, authorization, and usability.

Two sample codes (available at Github)

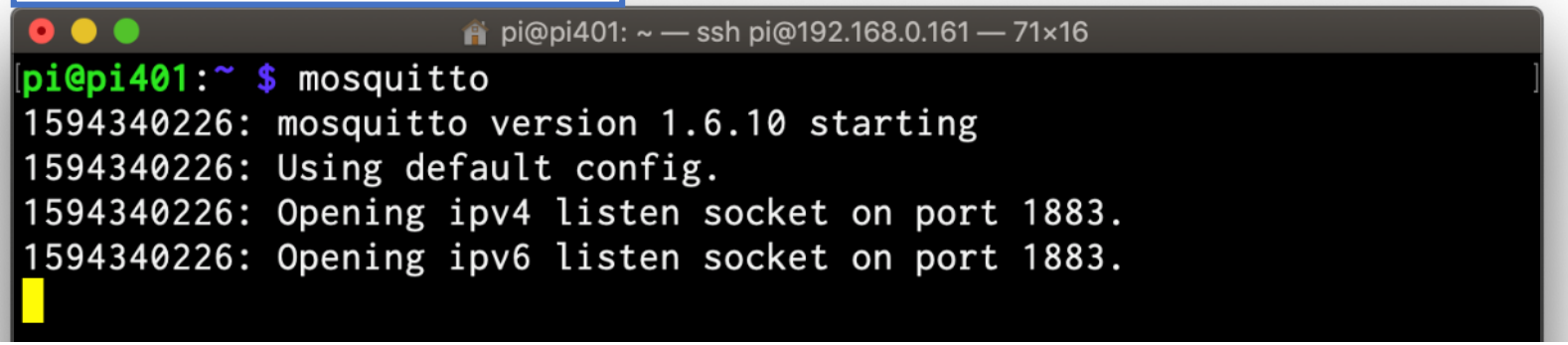
- Sample code 1: ncap.py (NCAP (Response Node))
  - Provide TEDS as a response of request message from applications
- Sample code 2: app.py (Application (Request Node))
  - Receive TEDS from NCAP by publishing a message with response_topic property
  - Time-out when Application does not receive TEDS soon
    - Application will notice unavailability of the NCAP

Application (app.py)



Step1: Run mosquitto (MQTT broker)

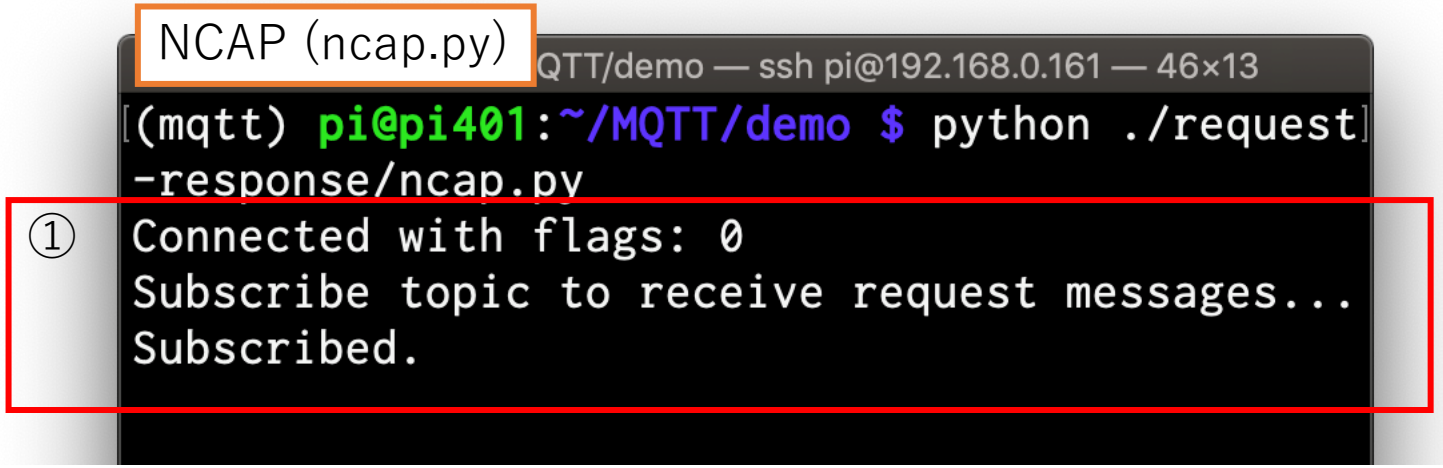
MQTT broker (mosquitto)



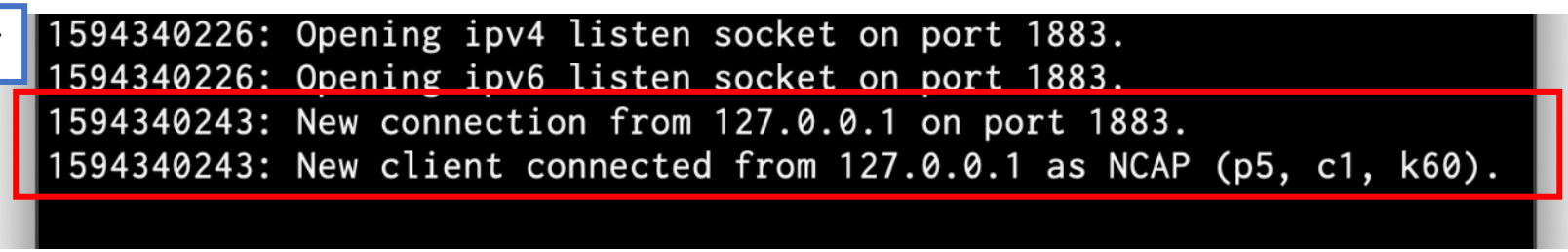
MQTT Broker

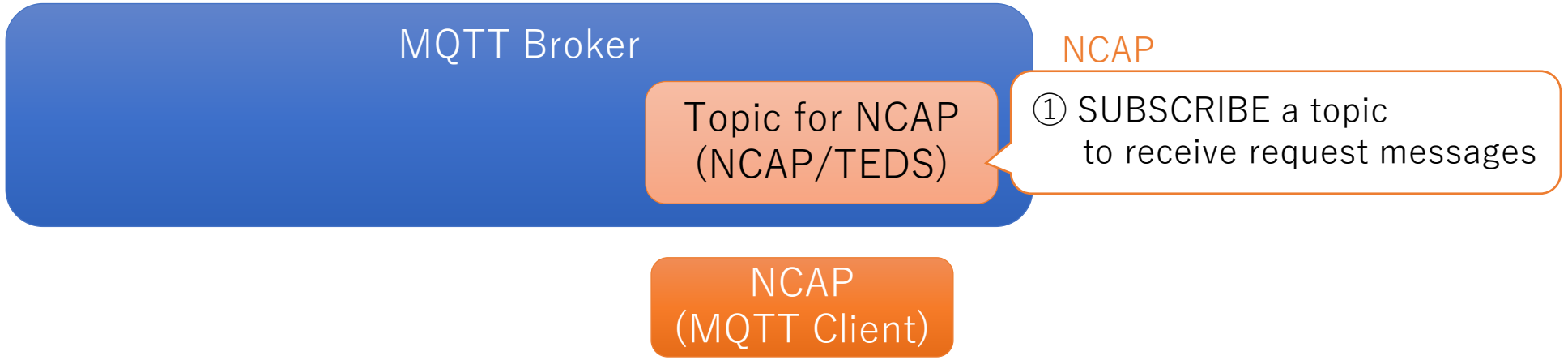
Step2: Run ncap.py

NCAP (ncap.py)



MQTT broker





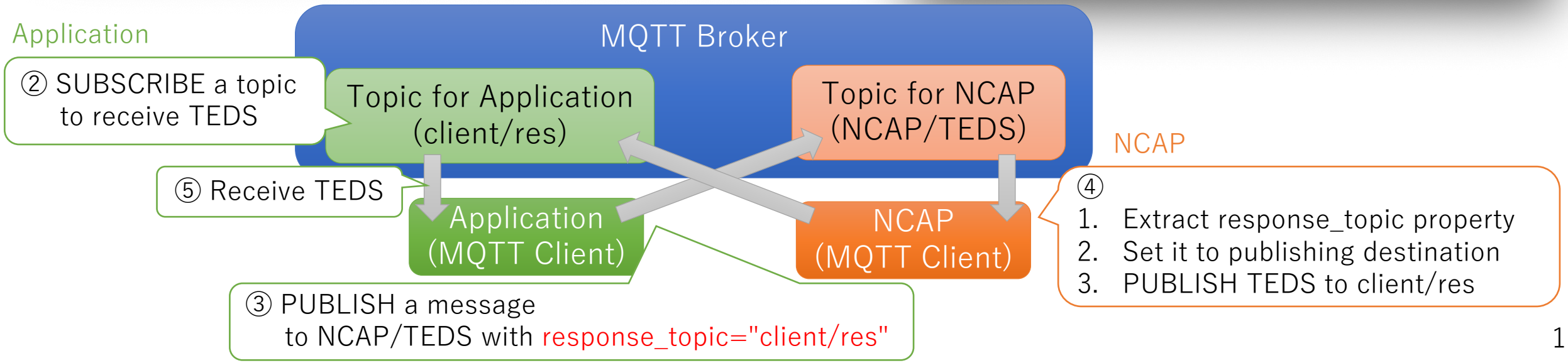
Step3: Run app.py

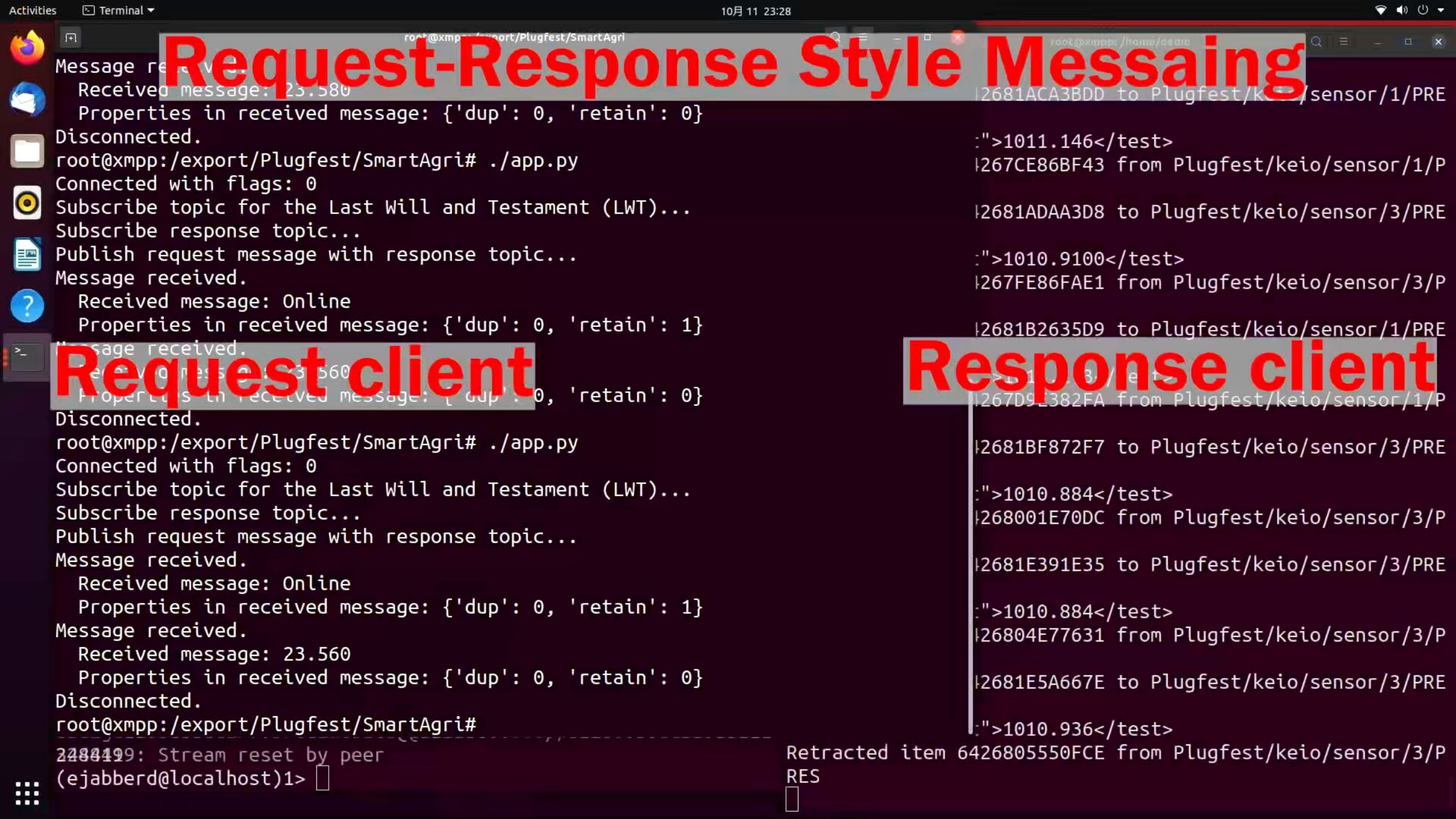
Application (app.py)

```
mo — ssh pi@192.168.0.161 — 56×11
(mqtt) pi@pi401: /MQTT/demo $ python ./request-response/
app.py
Connected with flags: 0
② Subscribe response topic...
③ Publish request message with response topic...
Subscribed.
⑤ Message received.
Received message: TEDS information from NCAP
Properties in received message: {'dup': 0, 'retain': 0}
Disconnected.
(mqtt) pi@pi401:~/MQTT/demo $
```

NCAP (ncap.py)

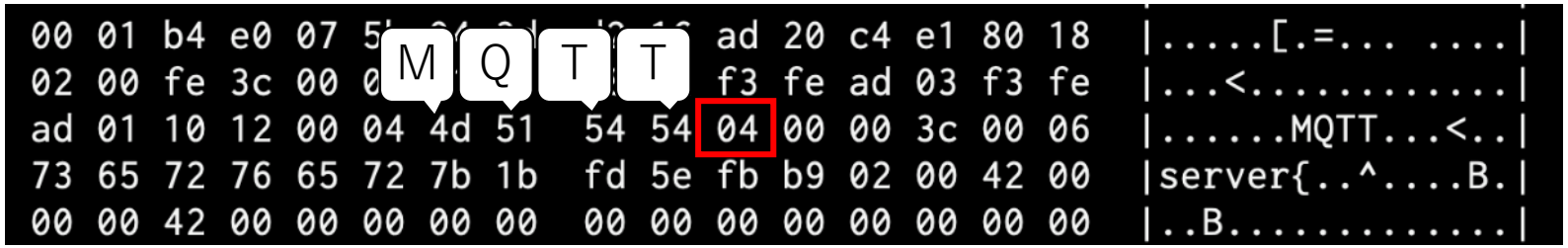
```
TT/demo — ssh pi@192.168.0.161 — 46×13
(mqtt) pi@pi401:~/MQTT/demo $ python ./request
-response/ncap.py
Connected with flags: 0
Subscribe topic to receive request messages...
Subscribed.
④ Message received.
Properties in received message: {'response_top
ic': ['client/res'], 'dup': 0, 'retain': 0}
Response topic in received message: client/res
Publish response message to response topic...
Published.
```



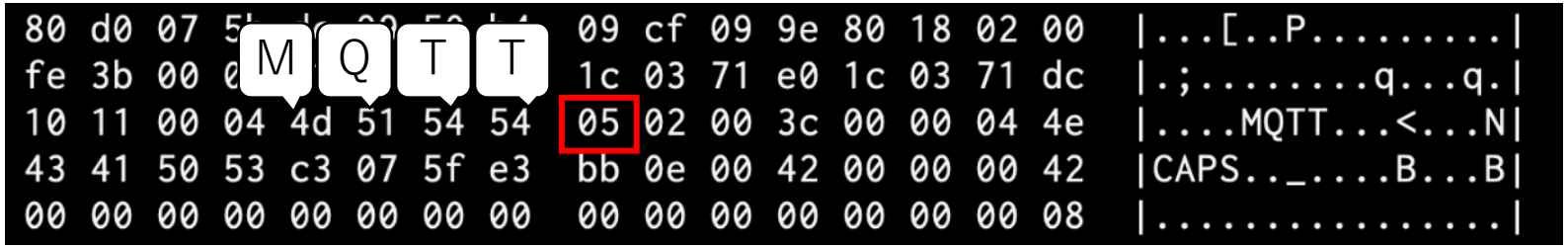

Request-Response Style Messaing
Request client
Response client
Activities
Terminal
10月 11 23:28
Received message: 23.580
Properties in received message: {'dup': 0, 'retain': 0}
Disconnected.
root@xmpp:/export/Plugfest/SmartAgri# ./app.py
Connected with flags: 0
Subscribe topic for the Last Will and Testament (LWT)...
Subscribe response topic...
Publish request message with response topic...
Message received.
Received message: Online
Properties in received message: {'dup': 0, 'retain': 1}
Message received.
Received message: 23.560
Properties in received message: {'dup': 0, 'retain': 0}
Disconnected.
root@xmpp:/export/Plugfest/SmartAgri# ./app.py
Connected with flags: 0
Subscribe topic for the Last Will and Testament (LWT)...
Subscribe response topic...
Publish request message with response topic...
Message received.
Received message: Online
Properties in received message: {'dup': 0, 'retain': 1}
Message received.
Received message: 23.560
Properties in received message: {'dup': 0, 'retain': 0}
Disconnected.
root@xmpp:/export/Plugfest/SmartAgri#
(ejabberd@localhost)1>
2681ACA3BDD to Plugfest/keio/sensor/1/PRE
">1011.146</test>
267CE86BF43 from Plugfest/keio/sensor/1/P
2681ADAA3D8 to Plugfest/keio/sensor/3/PRE
">1010.9100</test>
267FE86FAE1 from Plugfest/keio/sensor/3/P
2681B2635D9 to Plugfest/keio/sensor/1/PRE
267D9E382FA from Plugfest/keio/sensor/1/P
2681BF872F7 to Plugfest/keio/sensor/3/PRE
">1010.884</test>
268001E70DC from Plugfest/keio/sensor/3/P
2681E391E35 to Plugfest/keio/sensor/3/PRE
">1010.884</test>
26804E77631 from Plugfest/keio/sensor/3/P
2681E5A667E to Plugfest/keio/sensor/3/PRE
">1010.936</test>
Retracted item 6426805550FCE from Plugfest/keio/sensor/3/P
RES

How to confirm MQTT version?

- Mosquitto and gmqtt automatically downgrade to MQTTv3 when it receives MQTTv3 message
- If MQTTv3
  - "04" appears after "MQTT" in network packet



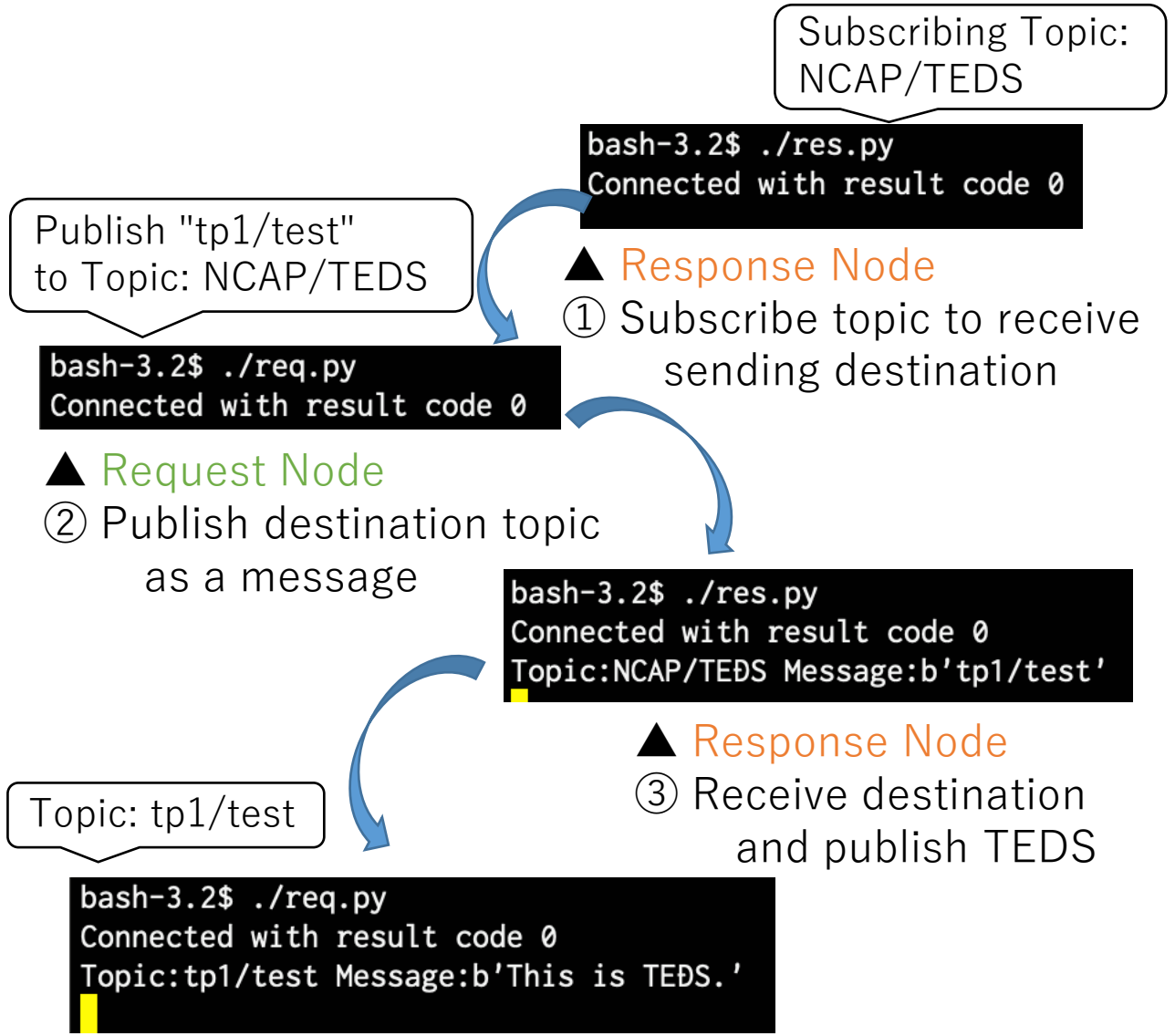
- If MQTTv5
  - "05" appears after "MQTT" in network packet



Sample codes (MQTTv3)

- Software Environment
  - MQTT Broker:
    - mosquitto 1.6.10
  - MQTT Clients:
    - Python3 with paho-mqtt 1.5.0
- Brief description
  - "Request Node" is the Application in the prior code
  - "Response Node" is the NCAP in the prior code
- Difference between MQTTv3 and MQTTv5
  - Request Node sends response_topic information as a normal message in MQTTv3

Message without property fields



Status of Open Source MQTTv5 support

- Broker server
- EMQ X (MQTT-SN, Erlang)
- flespi MQTT broker (Erlang)
- HiveMQ Community Edition (Java)
- JoramMQ (Java, Free eva. ready)
- KMQTT (Kotlin)
- mosquitto (Java)
- **Mosquitto (Thread safe, C)**
- mqttools (Python)
- VerneMQ (Erlang, OTP)

- Client
- HiveMQ MQTT Client (Java)
- M2Mqtt (C#)
- Machine Head (?, Clojure)
- Mosquitto (C, >1.6)
- MQTT-C (C)
- mqttools (Python)
- net-mqtt (Haskel)
- **paho MQTT (C/++, Java(scr.), Python, Go)**
- gMQTT (Python)
- wolfMQTT (C)
- OpenHAB MQTT binding (Java)

P1451.1.6 for 1451.0

- What is 1451?
  - Refer Kang and William's lecture on 1451.
- What are defined in P1451.1.6?
  - Topic naming rules to exchange IEEE1451 messages
  - Message data formats to use with MQTT
  - Design of states and processes for maintaining concerning protocols
  - IEEE1451 network service messages over MQTT
  - Simple and general style of MQTT message for the use of 1451 devices
  - Time synchronization
- Status of IEEE P1451.1.6
  - Documentation has already finished, but not revised according to the new 1451.0D4 document.
  - We will update the document according to the final version of P1451.0 document.

Implementation and Open Sources

- Implementations for Interoperability and simple operation check
- Need Raspberry-Pi and sensors (DHT11)
- All message formats of D-OP, D0C-OP, and D0-OP are supported.
- Only reading sample data and reading TEDS are supported according to the request.
- Data formats are extended to use with MQTT and from the perspective of natural definition.
- The system environment of this lecture uses NCAP with TIM and attached sensors on Raspberry-Pi and NCAP application as Node-RED.

- GitHub (World biggest software development platform)
  - Including sandboxes of testing codes and data
  - https://github.com/westlab/IEEE-P1451.1.6
- IEEE-SA OPEN
  - Primay site (recommended)
  - https://opensource.ieee.org/west/ieee-p1451.1.6
- Sorry both are too old…

Message data formats in .1.6

- D0-OP (Dot Zero Operation)
  - IEEE P1451.0-based (Dot Zero-based) Operation (D0-OP) uses the original 1451.0 binary network service messages. It is expected that D0-Message uses new 1451.0 standards message formats without any modification.
  - The suffix of the topic name shall start with '_1451.1.6/D0/' as SPFX and following TOM. LOC follows the string to be a completed topic name.
  - Strictly observe new 1451.0 standards.
- C-OP (CSV Operation)
  - Comma Separated Values (CVS) format can be used as an option. TOM is changed to 'D0C', such as '_1451.1.6/D0C/BLD1/FL3/ROOM3'.
  - Perfectly compatible with 1451.0 messages.
  - Simply solves and extends the field size limitation in D0-OP for acquiring useability
- D-OP (Direct Operation)
  - Direct format for enabling a simpler and more suitable format to use with MQTT.
  - Significant messages are only defined as D-OP.
  - Strictly observes OASIS MQTT guidelines and general MQTT use cases.

Message topics

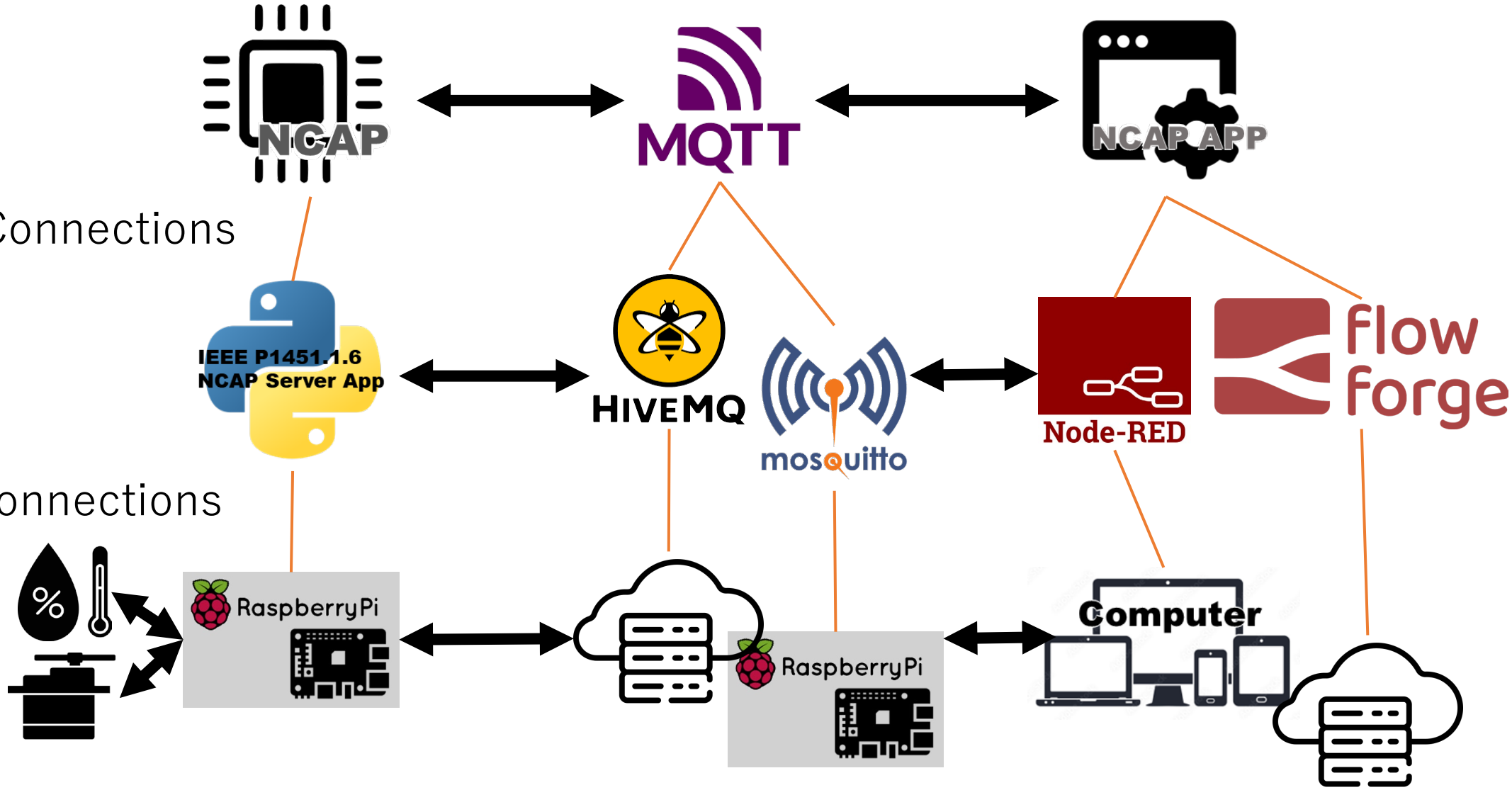
- Request-Response message is used
  - Request:
    - Data Request Topic: _1451.1.6/[D0 or D0C]/LOC of server/DATA/[TIM UUID or TIM NAME]
    - Data Request Message: The unique topic name to publish the sensor data, shortly [LOC of client]
    - NCAP server subscribes to this topic. SPFX, TOM, and concerning descriptors are not included.
  - Response:
    - Data Response Topic: [LOC of client]
    - Recommended Data Response Topic: _1451.1.6/[D0 or D0C]/[LOC of client]/RES/[TIM UUID or TIM NAME]
    - Data Response Message: Sensor Data
    - NCAP client subscribes to this topic.
- Publish-Subscribe communication
  - General communication style for MQTT
  - Dedicated topic: _1451.1.6/D/LOC/[TIM UUID or TIM NAME]
  - NCAP server publishes message to the topic.

Differences of message types

- NCAP announcement message is explained as an example
- D0-OP (binary)
  - IDL: Args:: UInt16 netSvc0101 {
    in Args::NetSvcType netSvcType = 1,
    in Args::NetSvcId netSvcId = 1,
    in Args::MsgType msgType = 3,
    in Args::UInt16 msgLength (2 bytes),
    in Args::UUID ncapId,
    in Args::_String ncapName (length = 16),
    in Args::AddressType addressType (1:IPv4, 2:IPv6),
    in Args::UInt8Array ncapAddress (length 4/16 bytes)
    };
- C-OP (Type dependent but size independent)
  - 1,1,3,ncapId,ncapName,addressType,ncapAddress

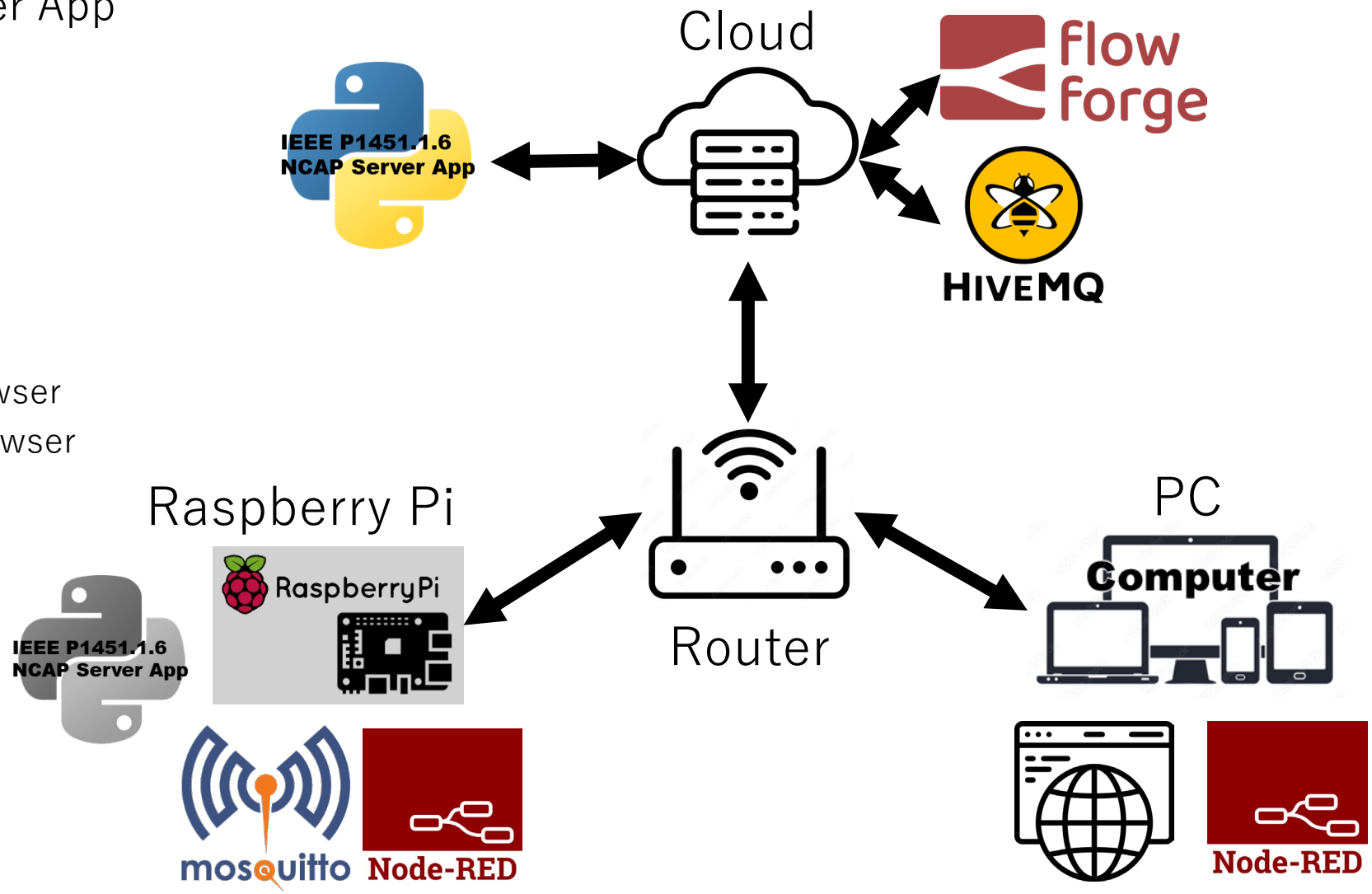
System configurations

- Logical Connections
- Software Connections
- Physical Connections



Typical network configurations

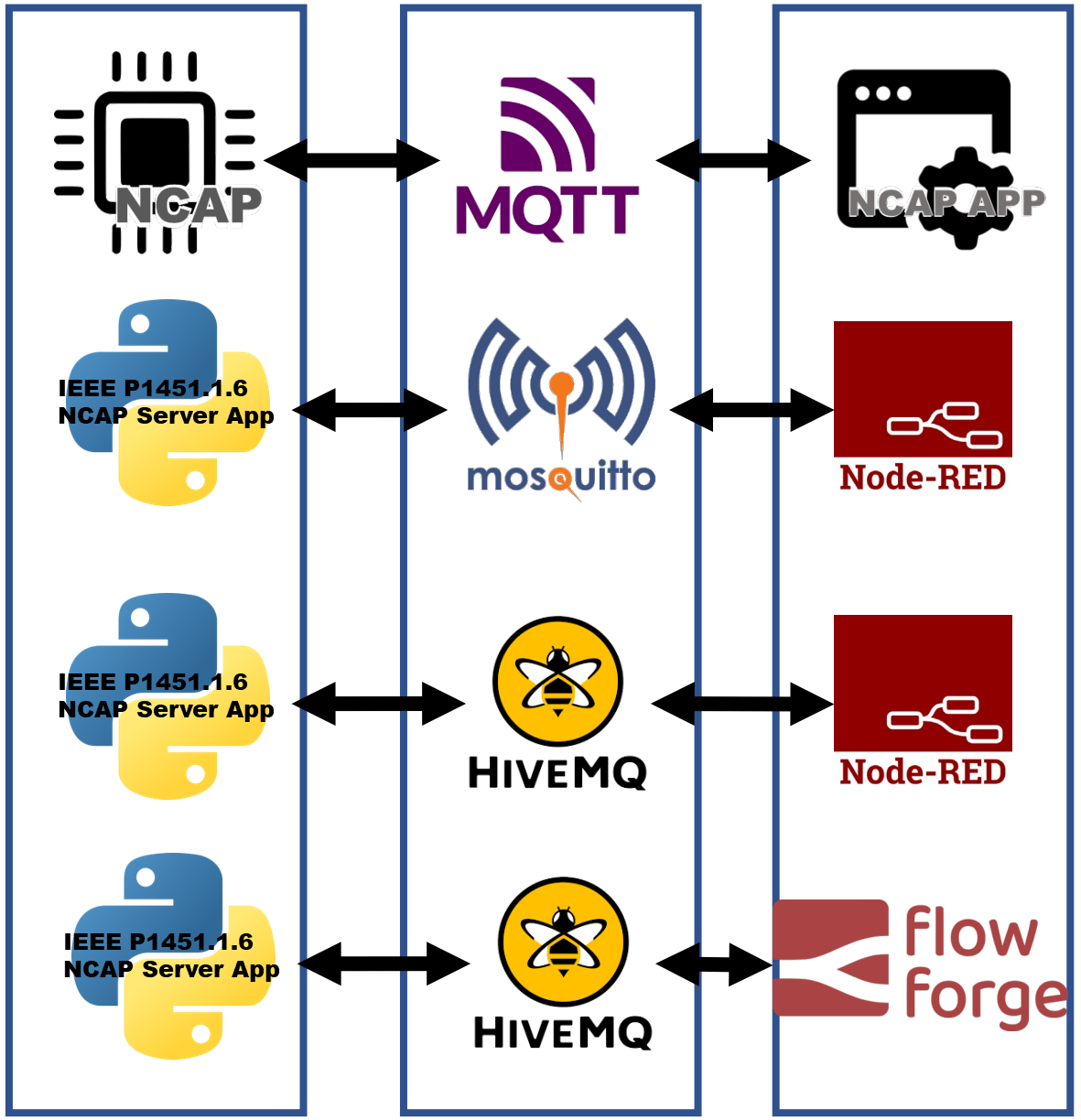
- Select one from NCAP, broker and App
  - IEEE P1451.1.6 NCAP Server App
  - MQTT broker
    - HIVE MQ (Cloud)
    - mosquitto (Raspberry Pi)
    - Keio@COE broker (option)
  - NCAP Application
    - Node-RED
      - Raspberry Pi and Pi browser
      - Raspberry Pi and PC browser
      - PC and PC browser
    - Flowforge
      - Cloud



Find your best combinations

- If you want to learn all configurations or if you only have Raspberry Pi
  - If you cannot access the Internet (you must configure sensors)
- If you do not have Raspberry Pi

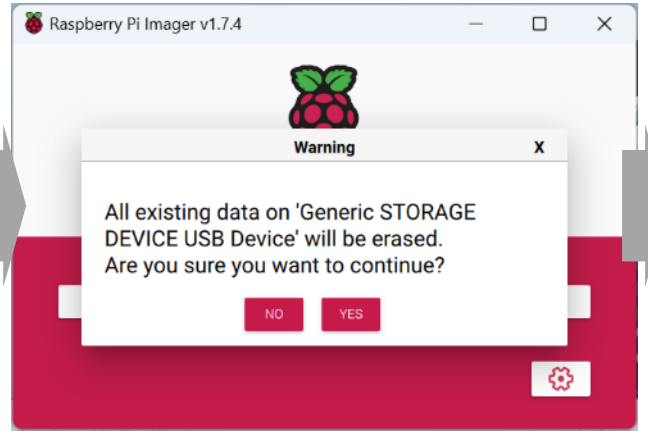
- If you do not want to install anything



Raspberry-Pi OS Installation

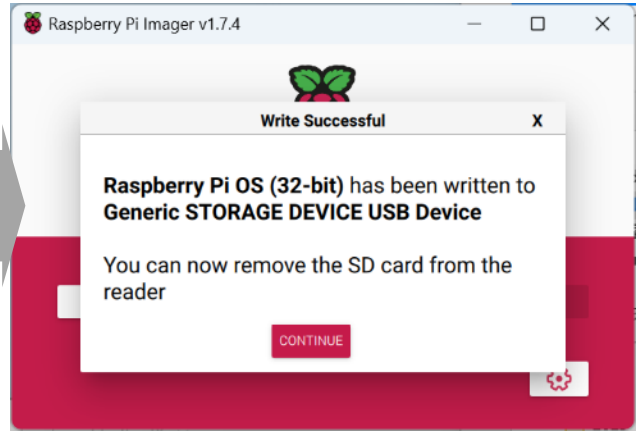
- https://www.raspberrypi.com/software/
- Download Pi Imager
- Select operatin system RASPBERRY PI OS(32-BIT) recommended
- Select Storage (Depends on your environments)
- Push Setting  to modify install options
  - You can set your hostname, default username/password, WiFi, locale, and enable SSH
- Push WRITE to burn OS image to your micro-SD card memory
- Check configuration menu to set your default login, password, and other settings

- Select Raspberry Pi OS 32-BIT
- Select Generic Storage Device USB



- Confirm your storage device is erased

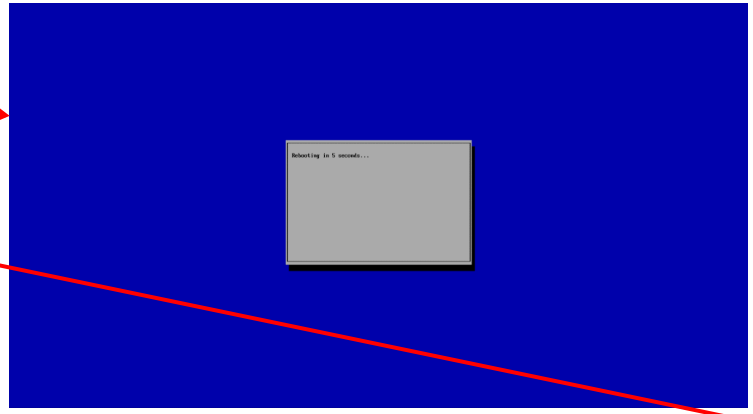
- Wait OS installation



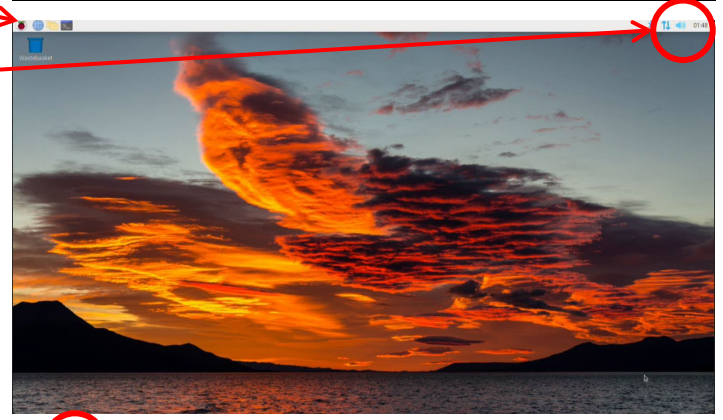
- Completed

Start up your Raspberry Pi with micro-SD card

- First boot for set-up
- opening screen

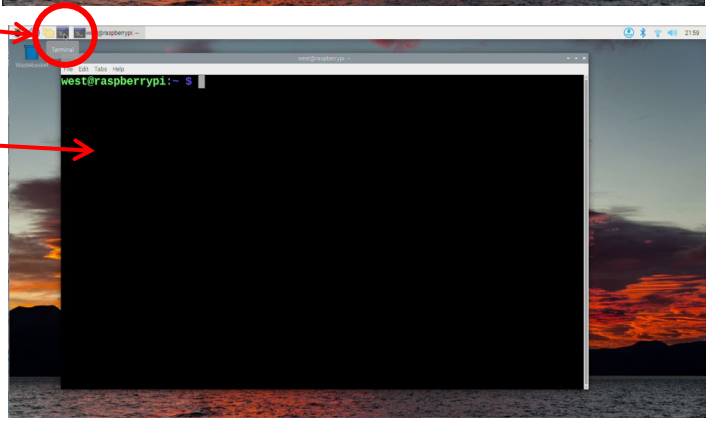


- Verify the configured network connection is established
  - Connect via WiFi or wired LAN



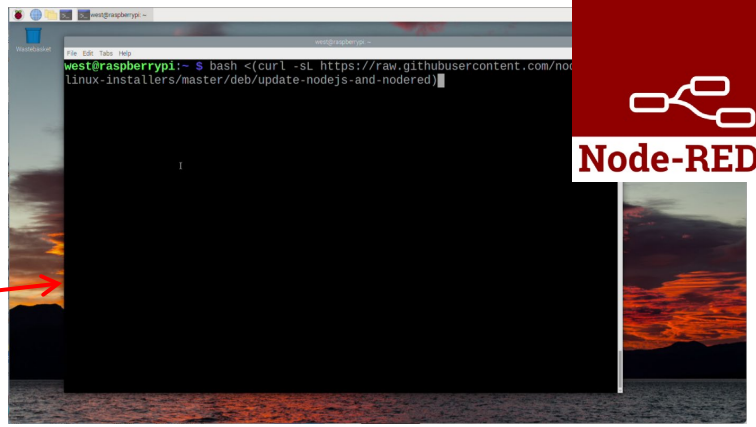
- Open Terminal

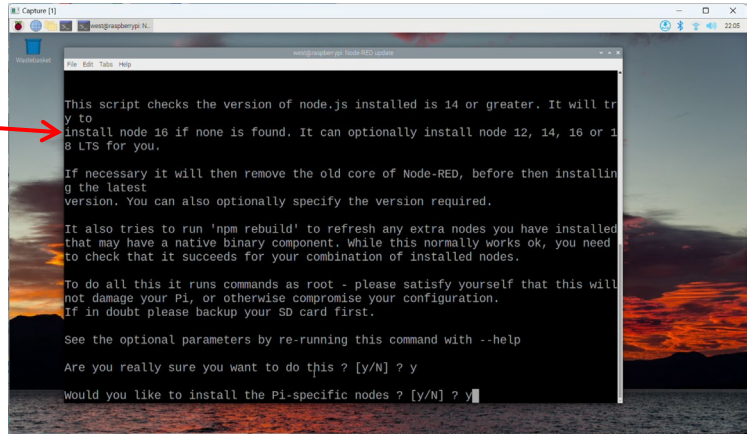
- Shell starts with the username and machine name you set.

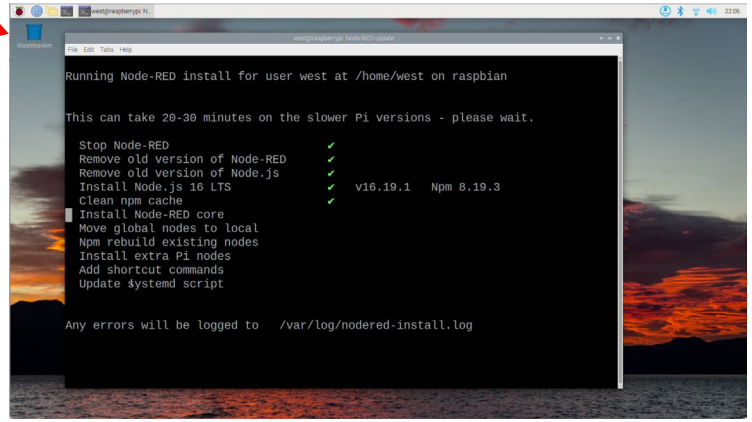


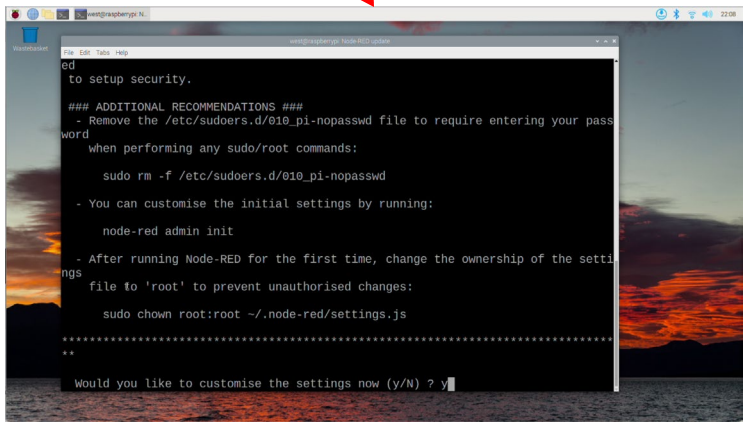
Install node-red

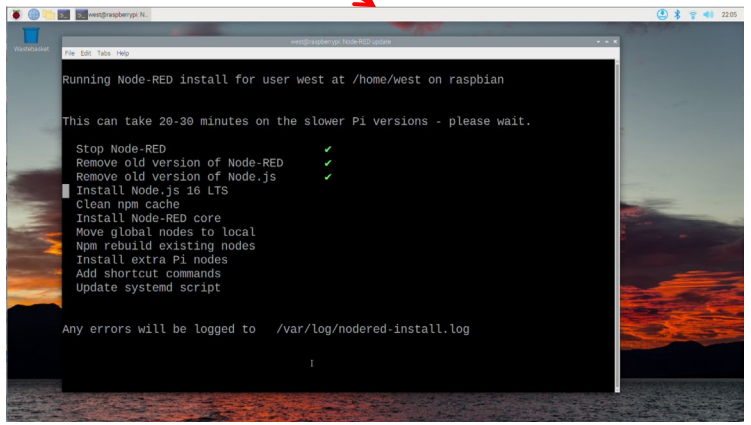
- Input the following command as one-liner command
  - bash <(curl -sL https://raw.githubusercontent.com/node-red/linux-installers/master/deb/update-nodejs-and-nodered)
- Press y twice
- Installation in progress
- Press y for finalize settings





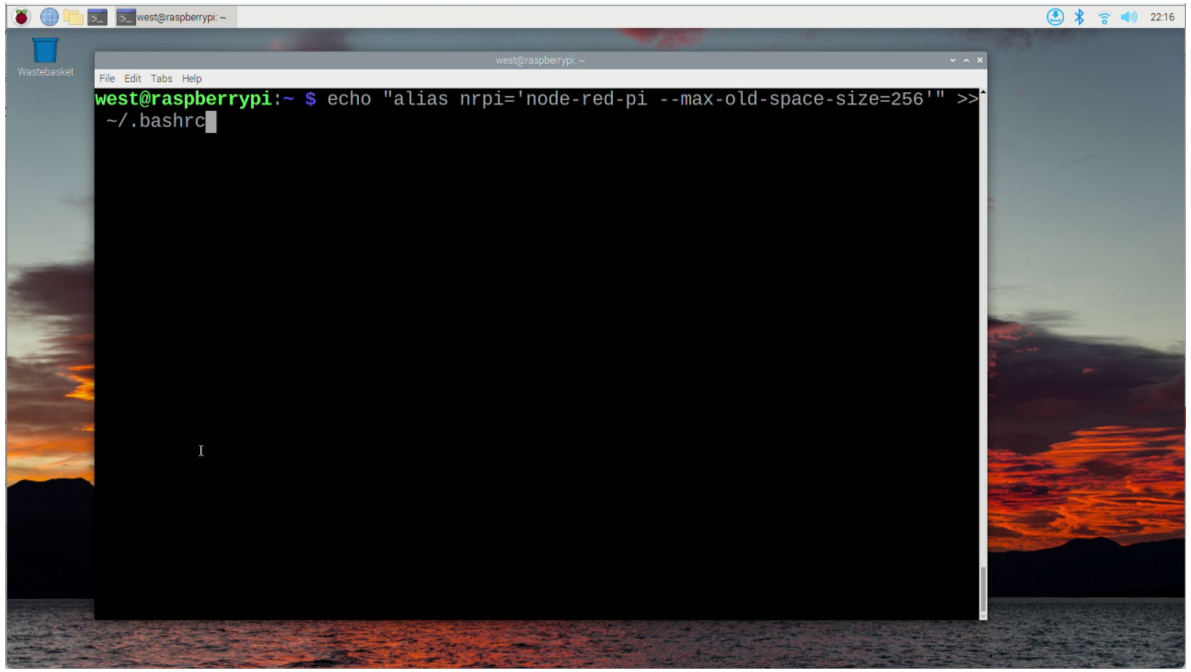


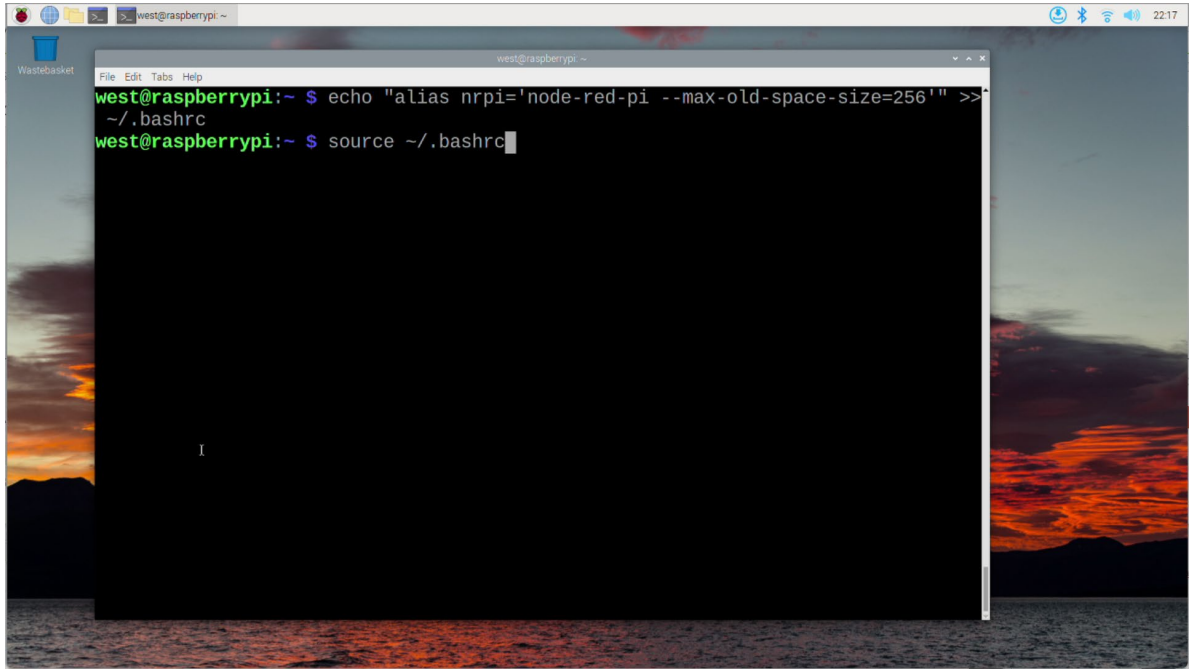




Complete Node-RED installation

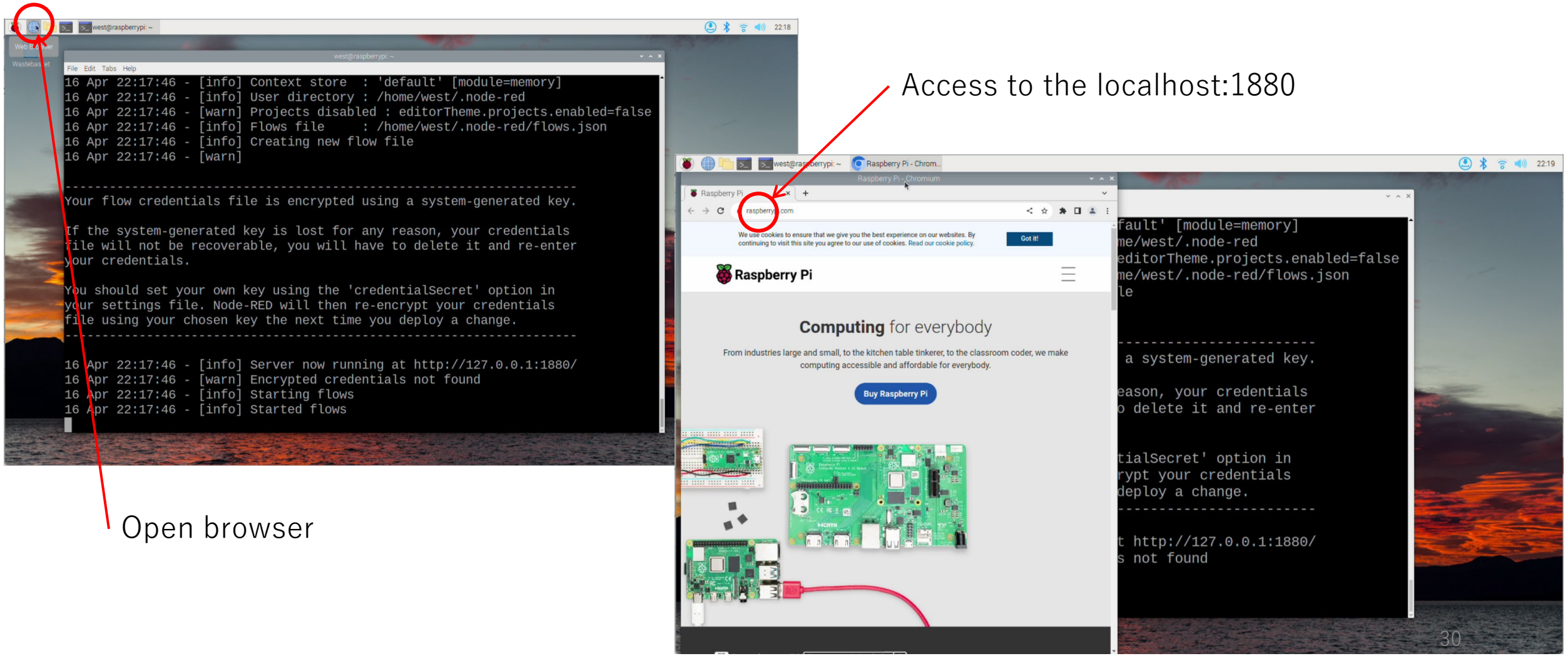
- Type and enter the following commands (only first time)
  - node-red-pi --max-old-space-size=256 ⏎
  - echo “alias nrpi node-red-pi --max-old-space-size=256” >> ~/.bashrc ⏎
  - source ~/.bashrc ⏎
- Run Node-RED
  - node-red ⏎





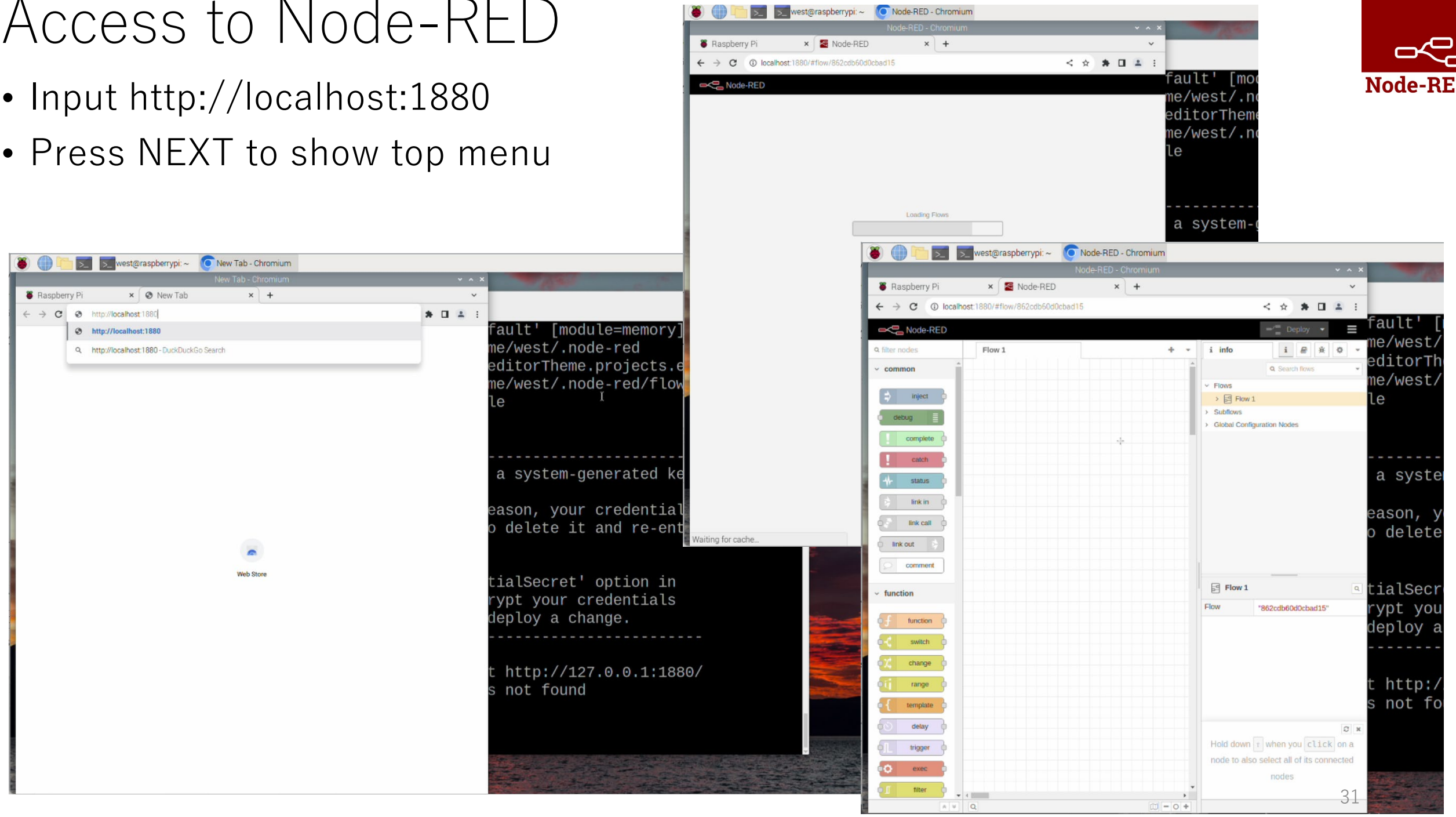
Run your browser to access Node-RED

- Node-RED is a web server for IoT design center
- Access to http://localhost:1880 or http://127.0.0.1:1880



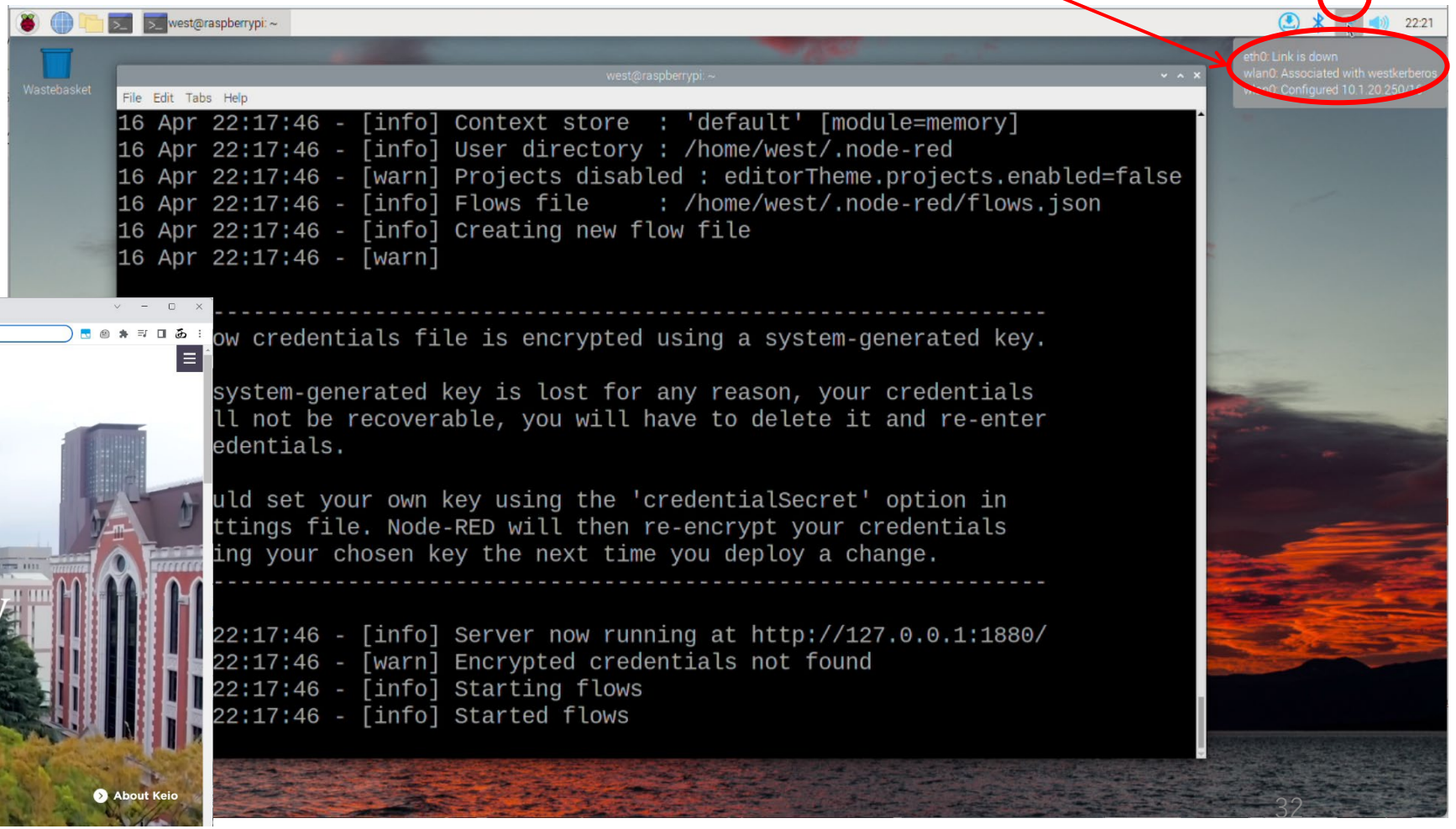
Access to Node-RED

- Input http://localhost:1880
- Press NEXT to show top menu

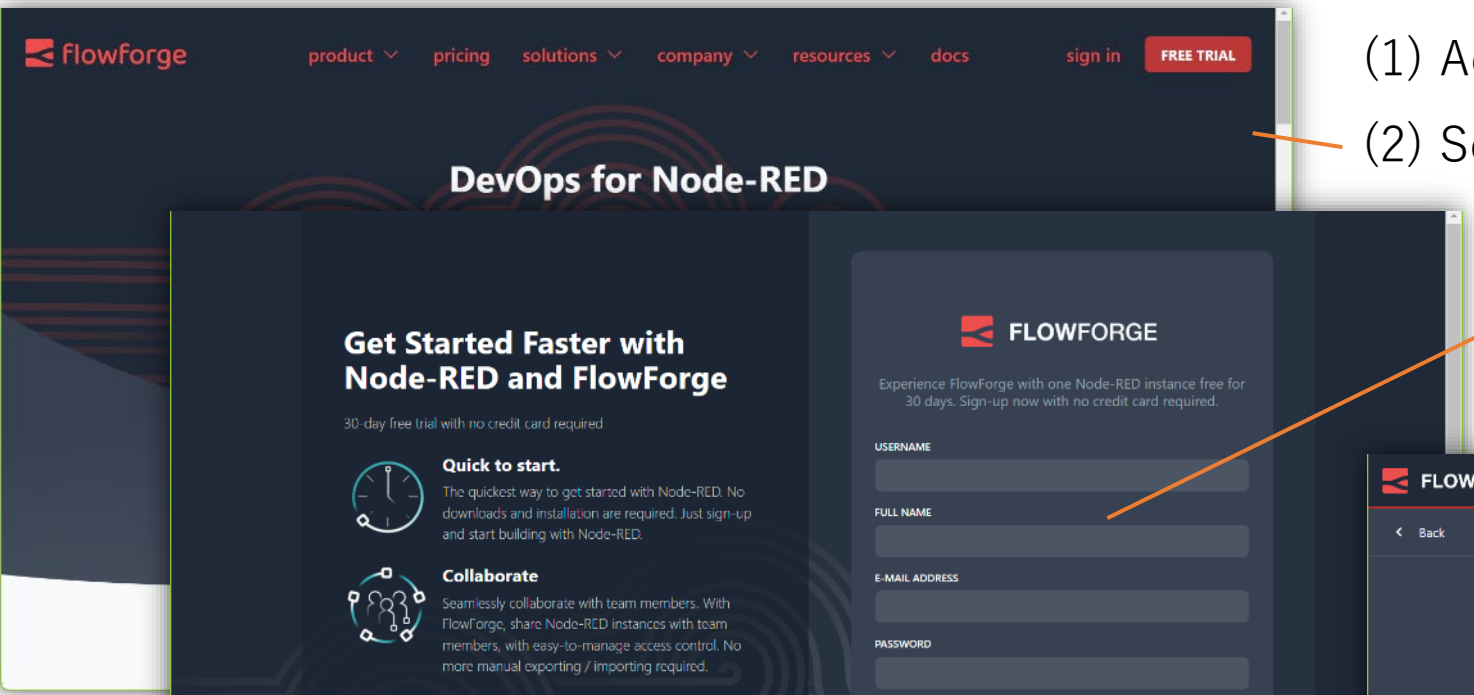


Access from your PC

- Check IP address by putting your mouse curser on the taskbar's network icon
- In this case the IP address of raspberry Pi is 10.1.20.250
- Access the IP:1880 from your PC



Use Flowforge (option)



(1) Access to https://flowforge.com

(2) Select FREE TRIAL

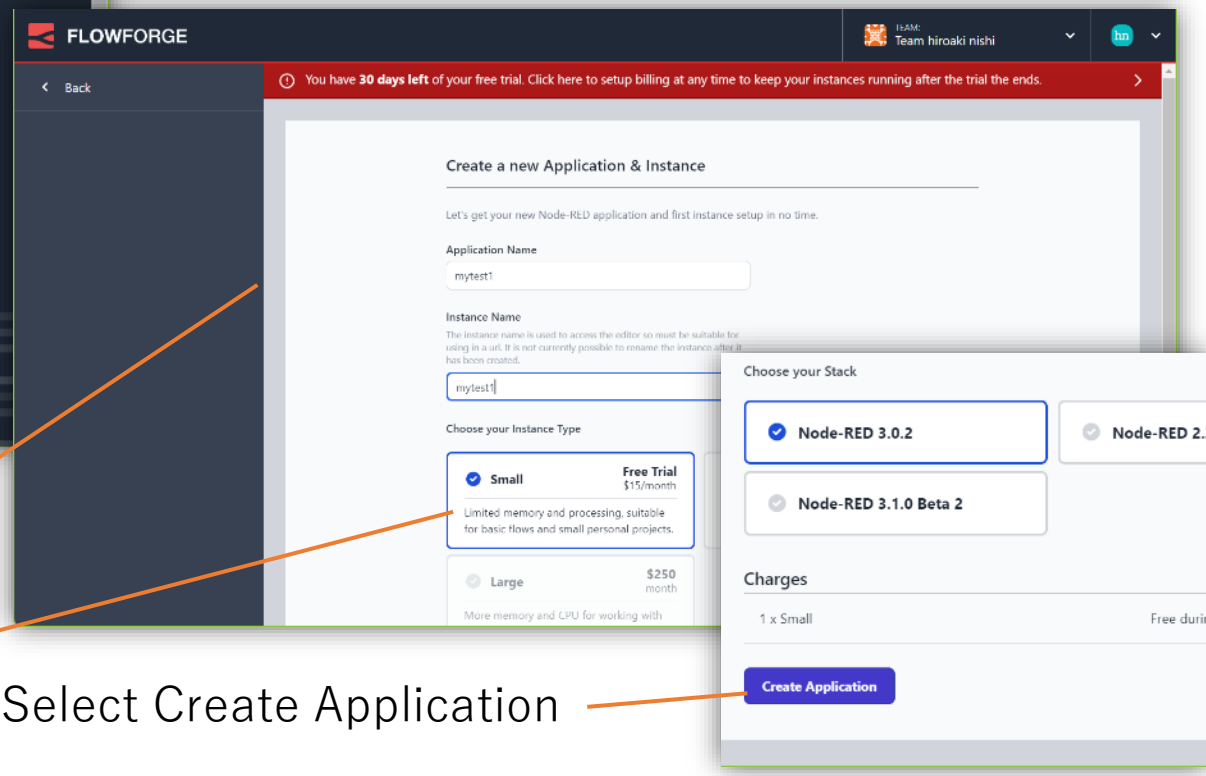
(3) Input required forms
USERNAME: (your username)
FULL NAME: (your name)
E-MAIL ADDRESS: (your e-mail address)
PASSWORD: パスワード(**your new password for this site**)

(4) Respond to the email sent to your registered address

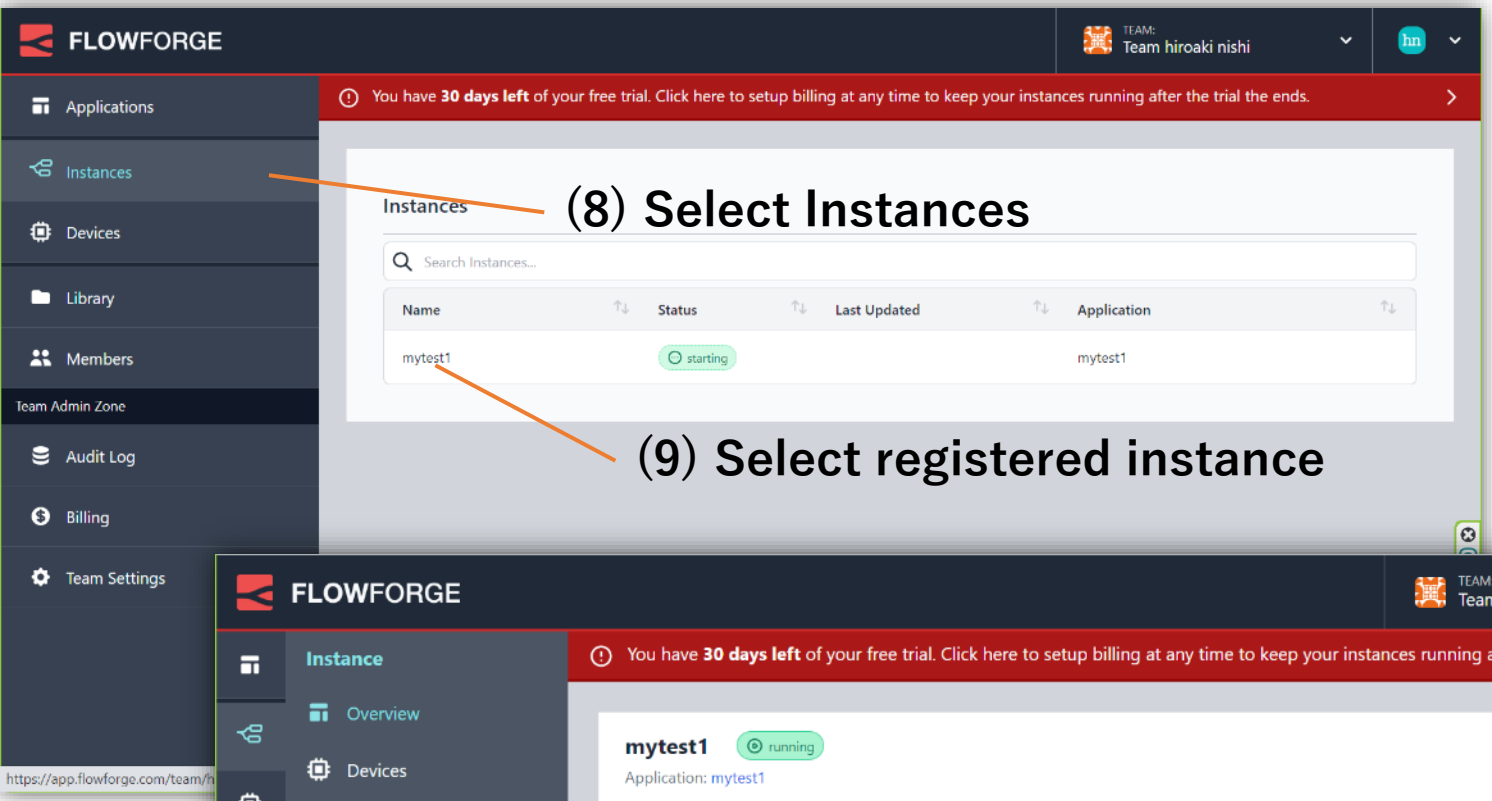
(5) Input Application Name and Instance Name

(6) Select Small Free Trial

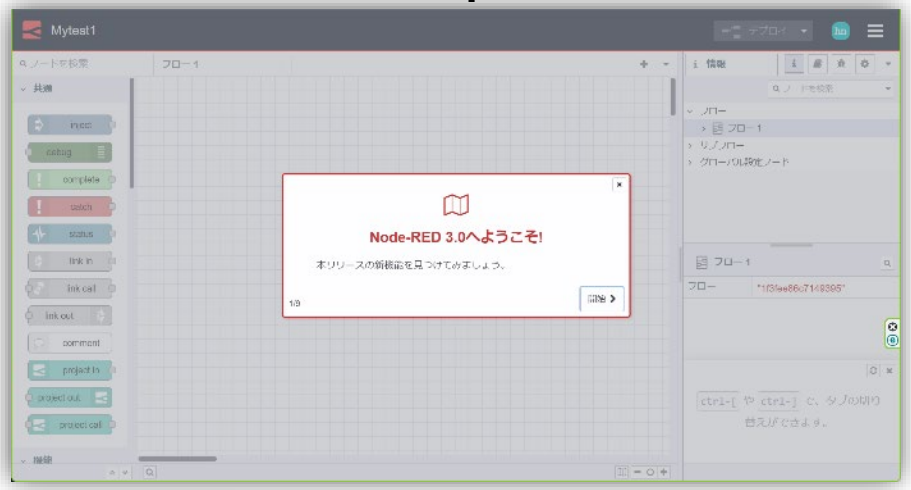
(7)Select Create Application

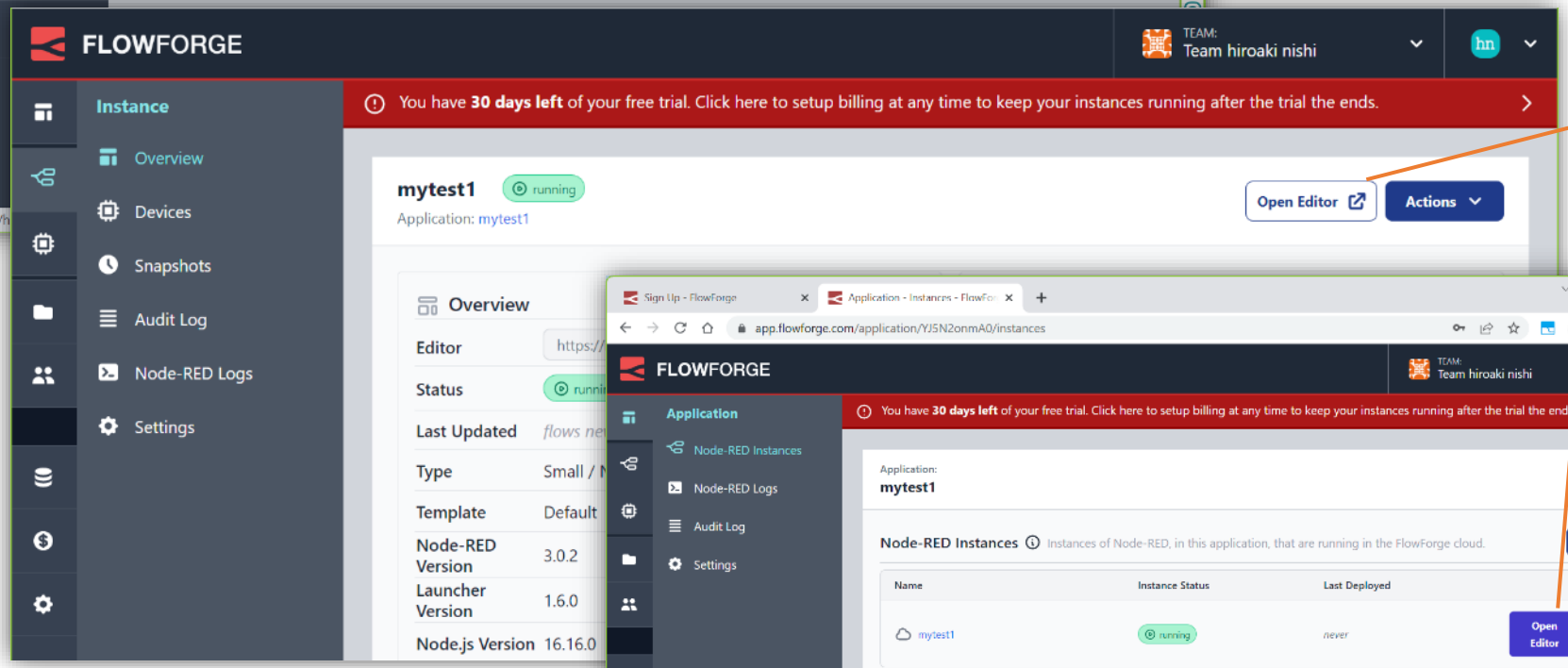


Set-up Flowforge



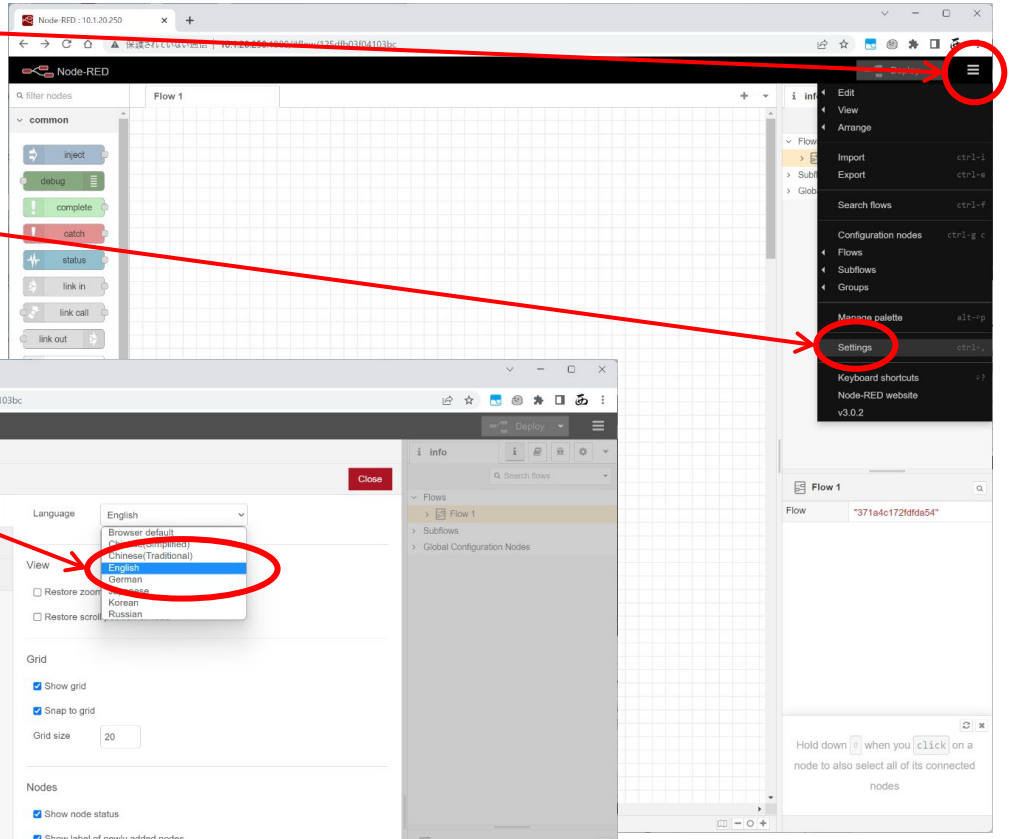
(12) Node-RED will open

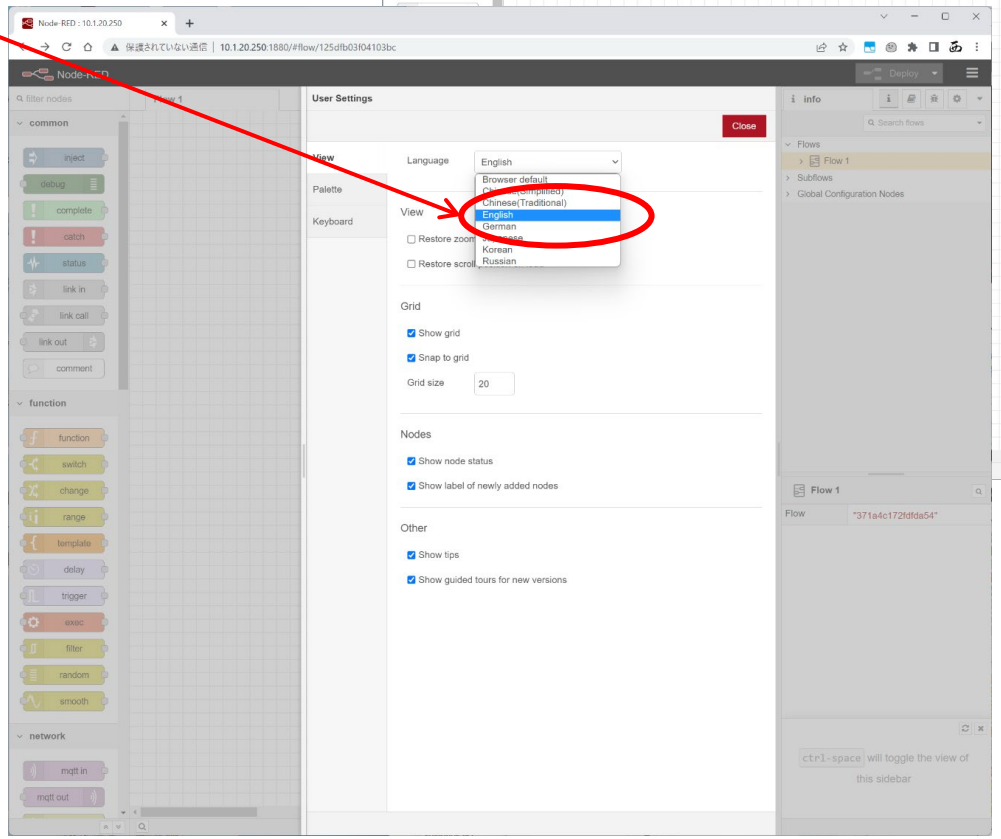


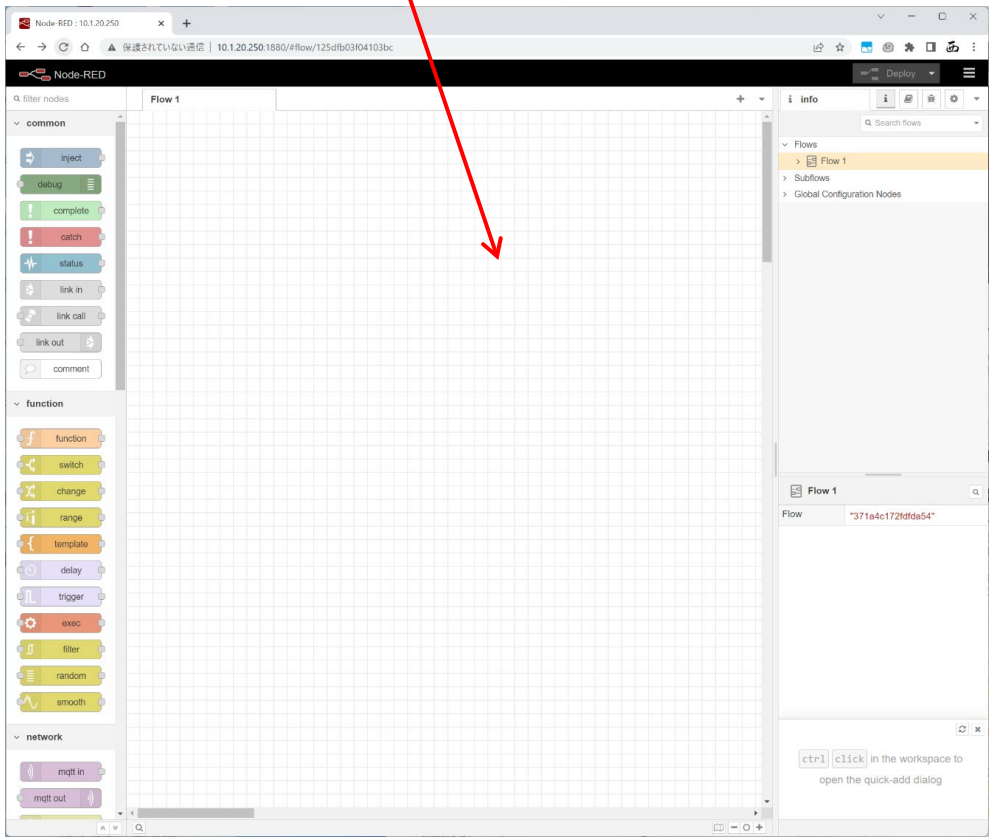


Change Node-RED language

- Press three-bar icon on the top right
- Press Settings
- Select language
- Working window





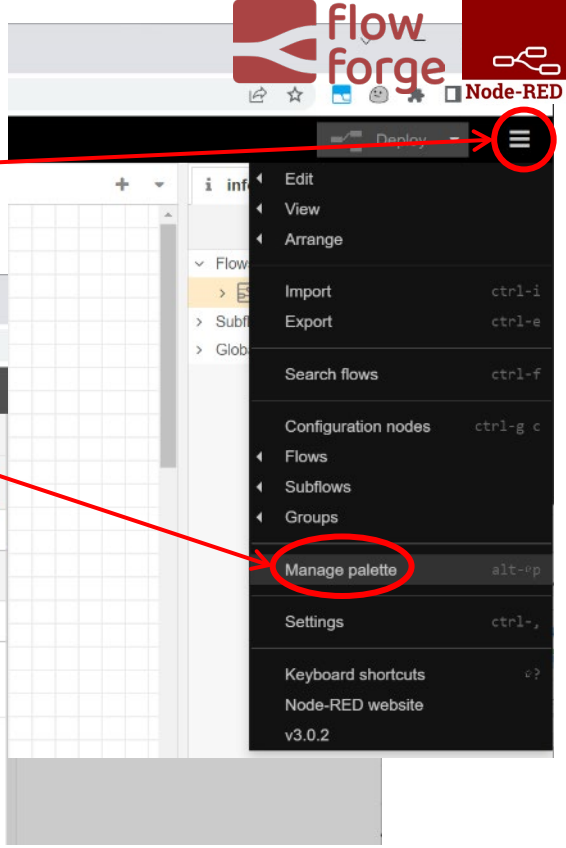


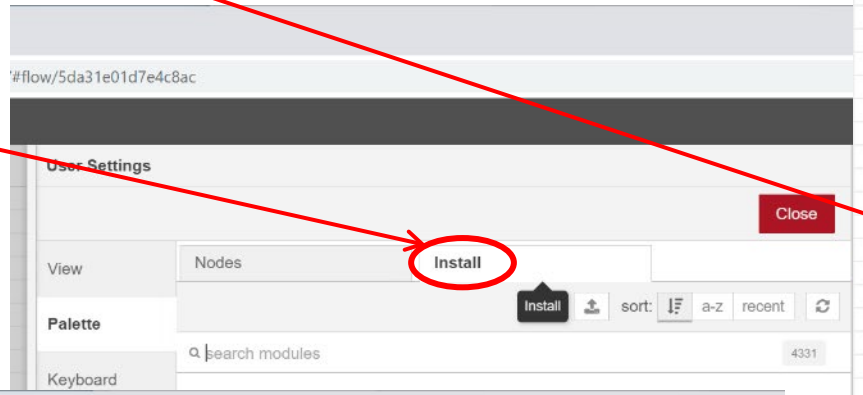
Design application on Node-RED

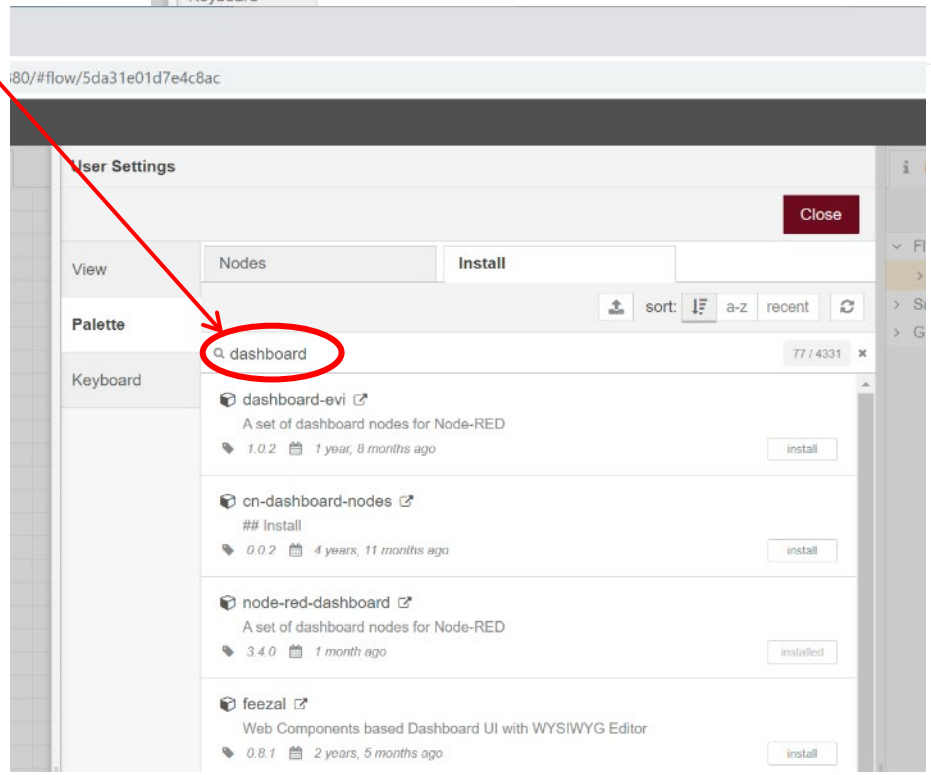
- Why Node-RED
  - Low-Code design
    - Do not type codes, but put nodes and connect them to design data flows
  - Many IoT interfaces
    - Dashboard
    - Cloud connections
    - Raspberry Pi connections
    - MQTT/HTTP/REST etc. etc.
    - Speech, Notification, email, SNS, etc. etc.
    - More than 4,000 add-ons are available

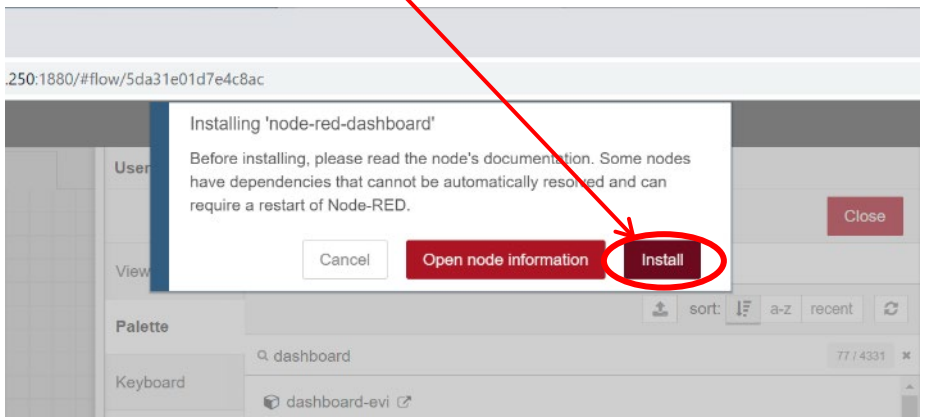
Dashboard installation

- Press Menu icon and Manage pallet
- Select Install tab
- Input dashboard in search window
- Press install



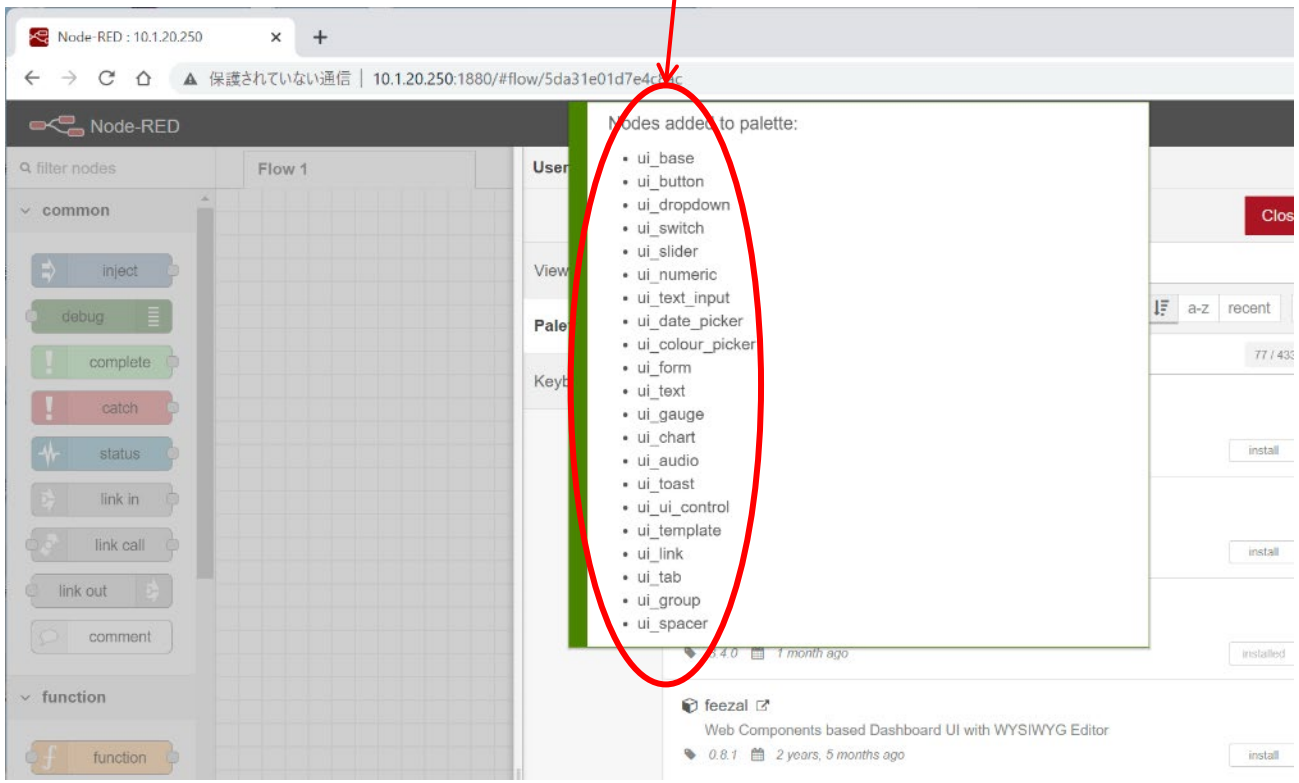


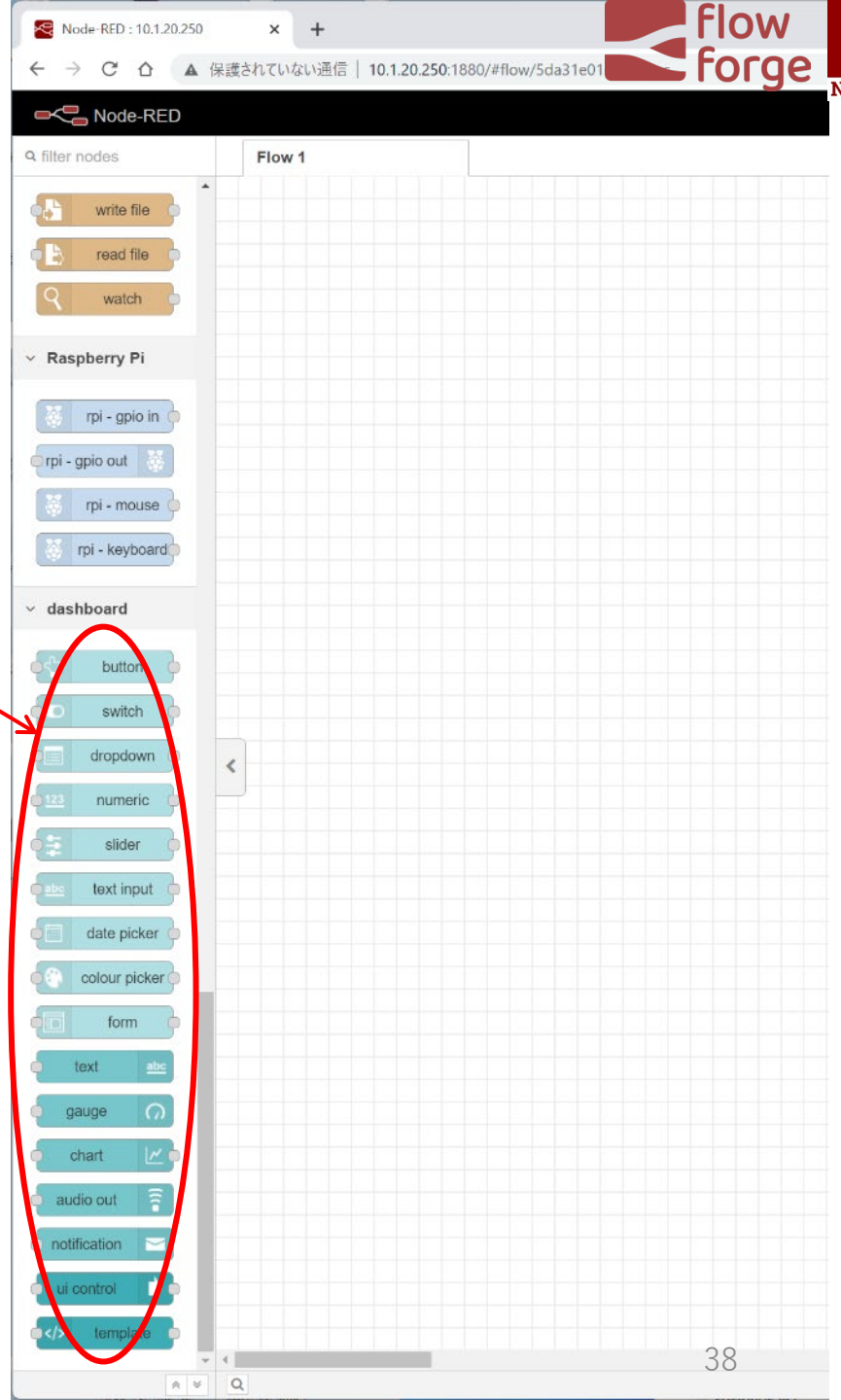




Dashboard nodes

- Installed nodes for dashboard

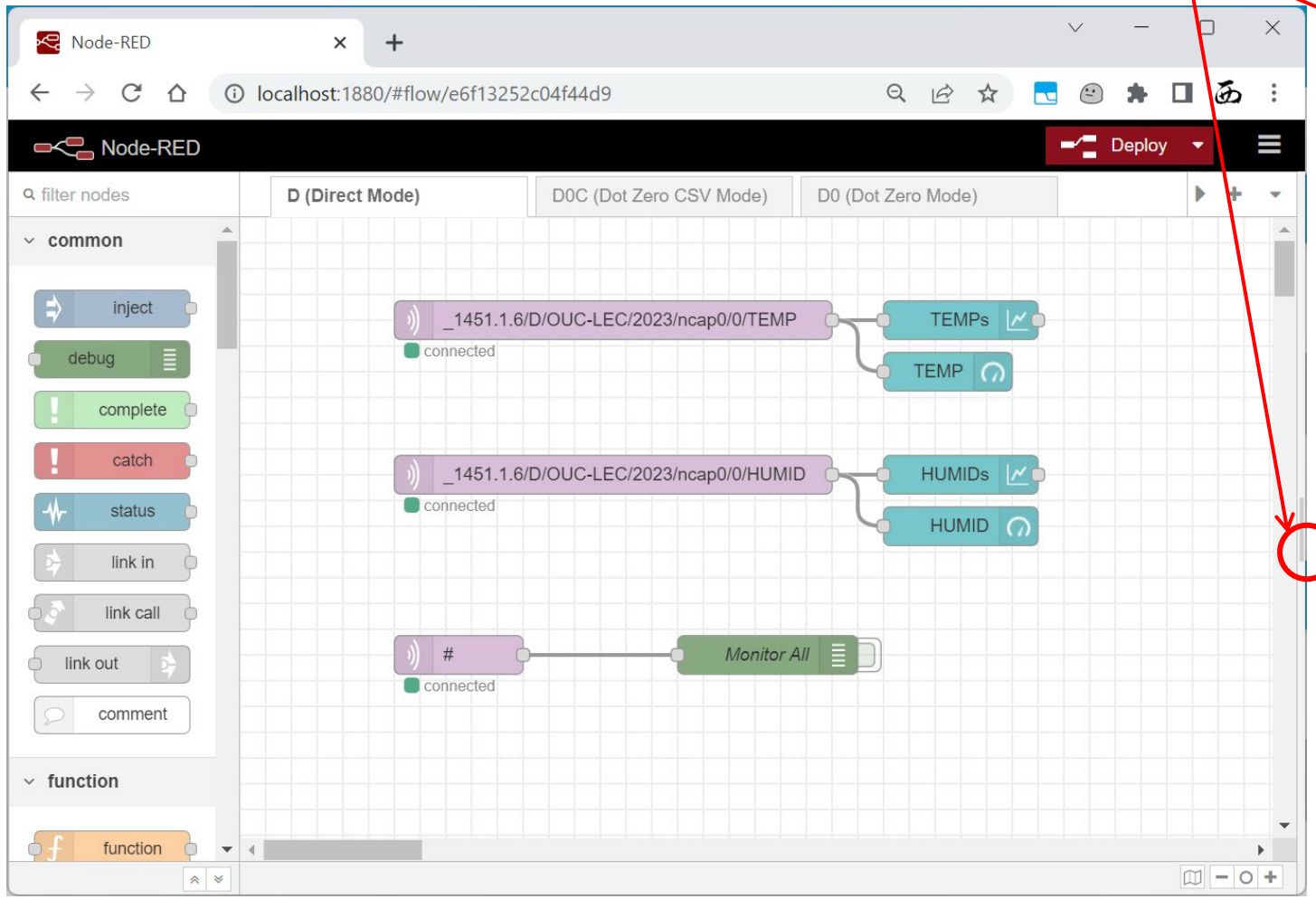




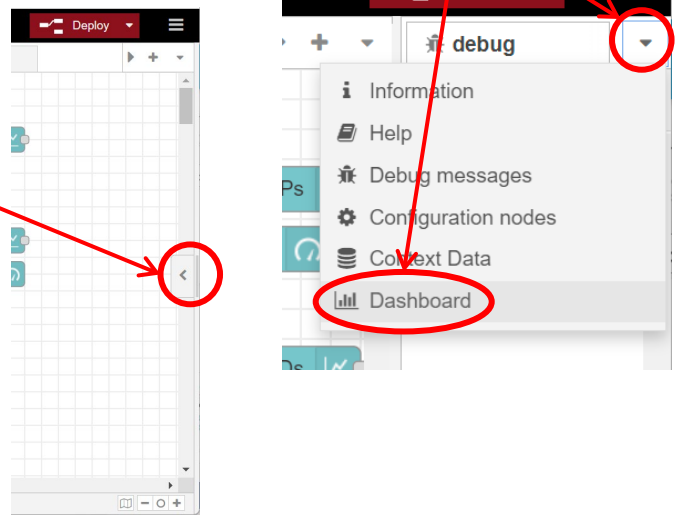
Direct Mode

- NCAP publish sensor data and subscribe them.
- Visualize the data

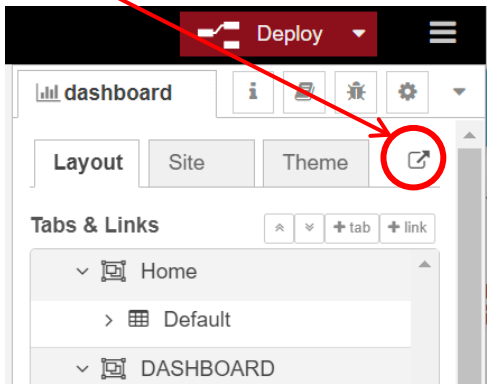
(1) Push sidebar open switch



(2) Push open menu and Dashboard

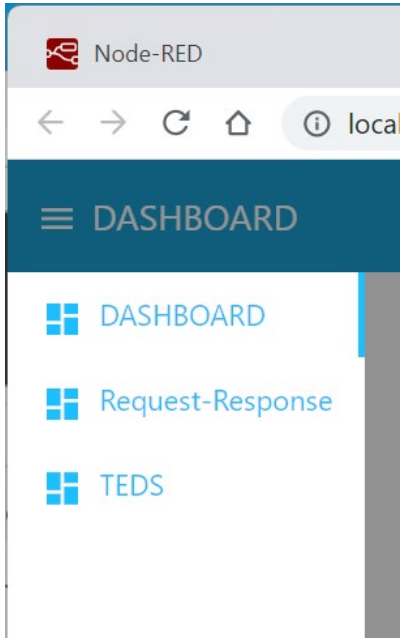


(3) Push open Dashboard link

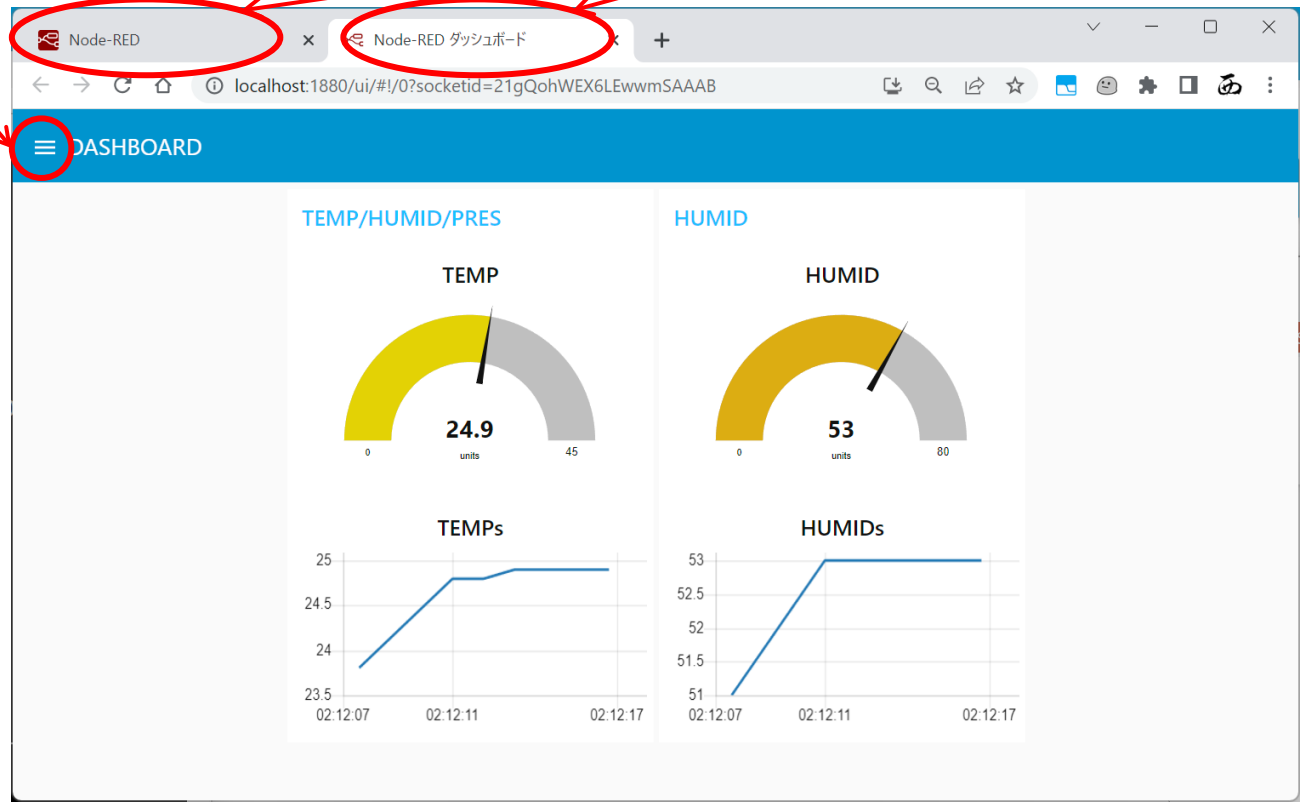


Direct Mode Dashboard

- The dashboard is fully customizable from Node-RED menu.
- Push menu to select a tab.
  - DASHBOARD: D0 dashboard
  - Request-Response: D0C, D0 sensor/actuator dashboard
  - TEDS: D0C, D0 TEDS dashboard



Push to select design screen
or dashboard screen

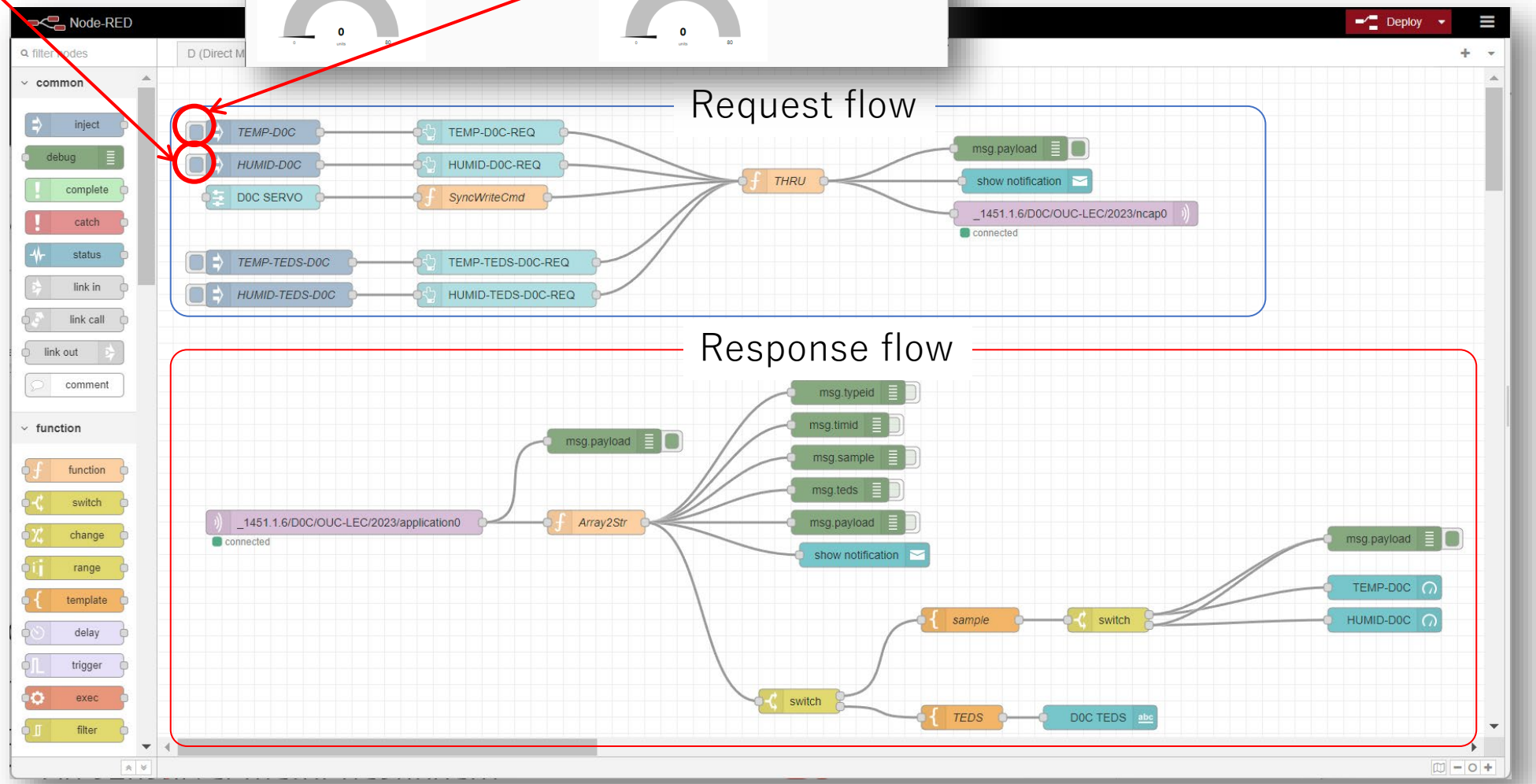


D0C Operation

Push HUMID to get current humidity

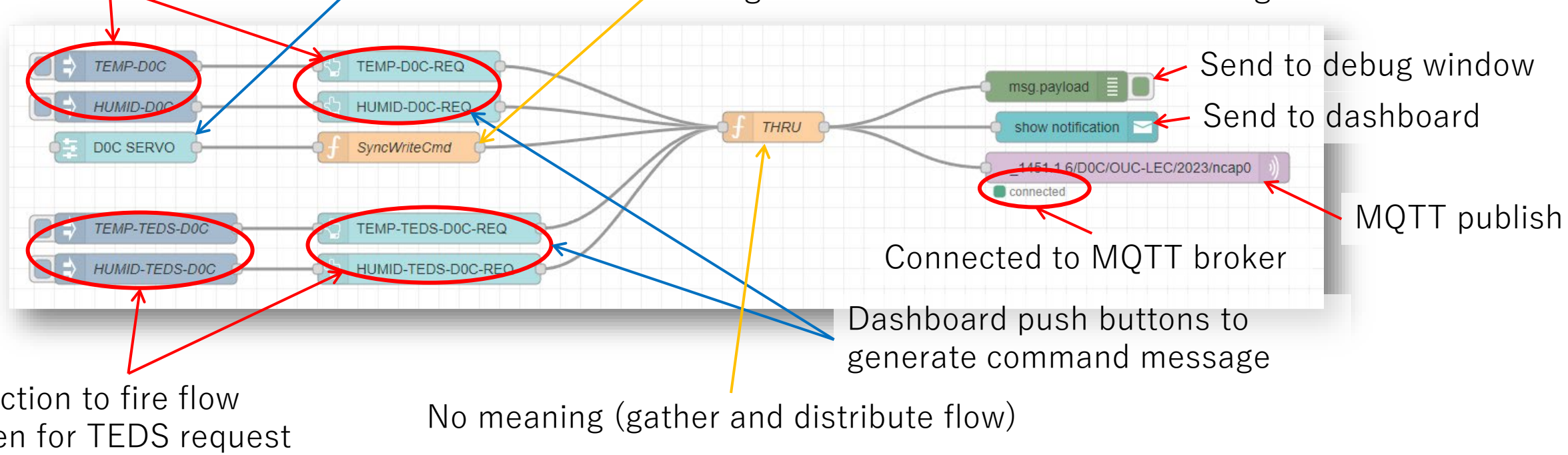
Slide SERVO to change servo angle

Push TEMP to get current temperature

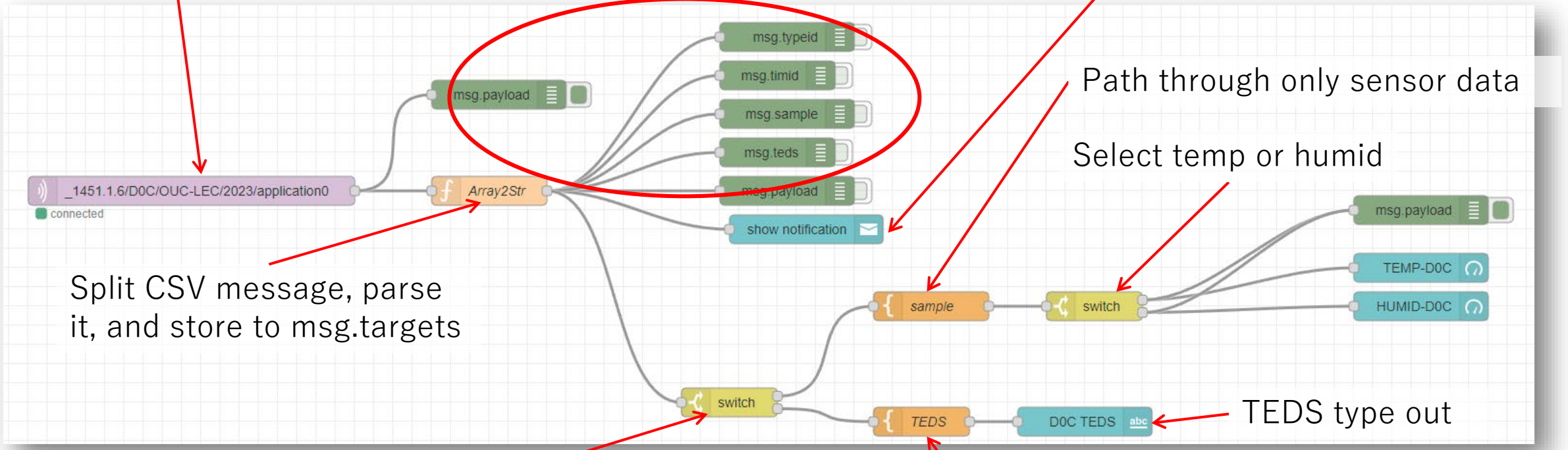


Request flow for D0C

- Click the nodes to show or change the properties of the node.



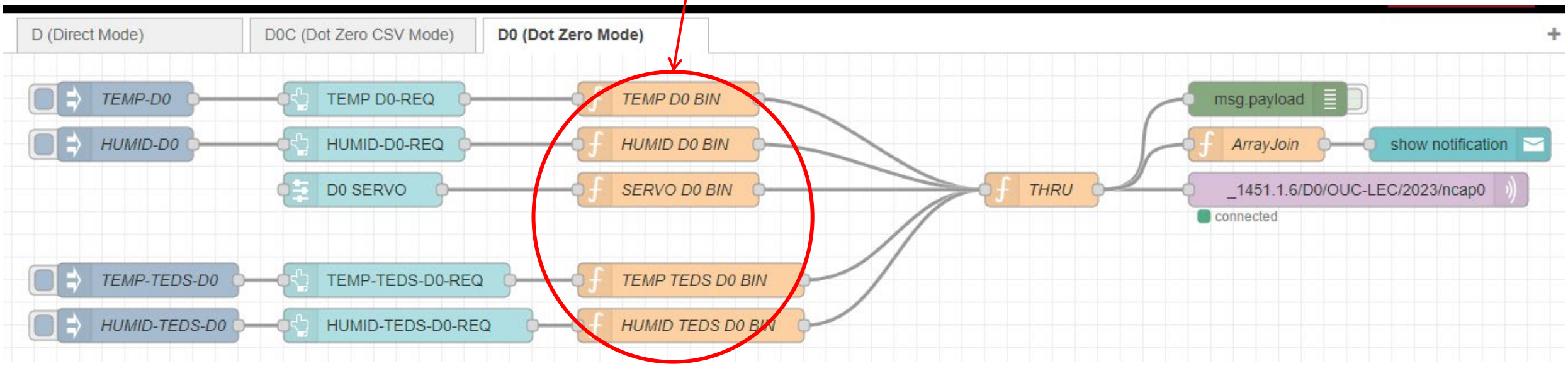
Request flow for D0C



D0 operation

- It is quite similar to D0C operations.
- The generated data is 1451.0 native binary message, and it is configured as a binary number array.

- The biggest difference to the D0C is functions for creating messages.

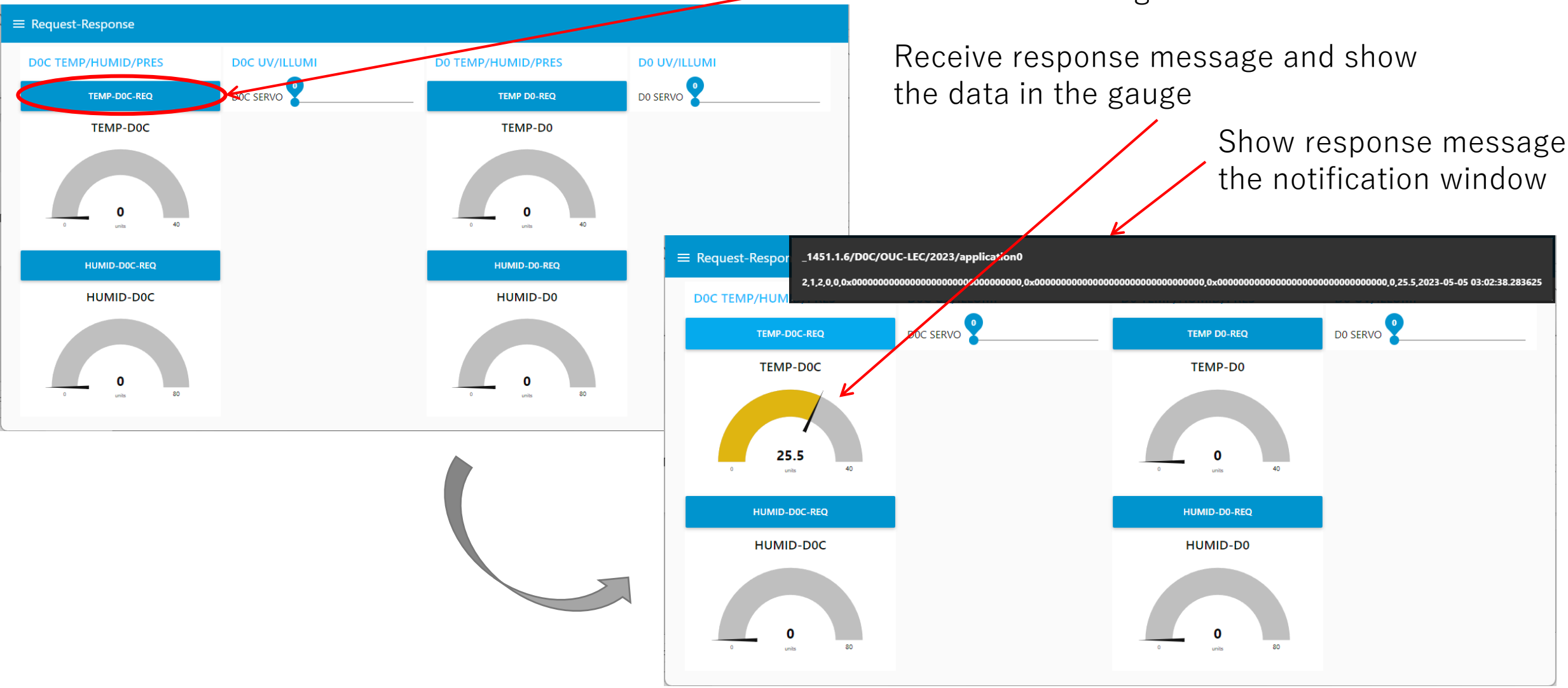


Request-Response Dashboard

Push temperature request to send read request command message

Receive response message and show the data in the gauge

Show response message in the notification window



Summary

- Learnt 1451.0 read/write messages
- Accessed NCAP with TIM and Transducers
  - Temperature Sensor
  - Humidity Sensor
  - Servo Motor
- Designed NCAP Application as a visualizer of sensor data
- Accessed TEDS of Transducers
- All was achieved by using Raspberry Pi and Node-RED

- NCAP is designed by using Python and is available on GitHub
- 1451.0 and 1451.1.6 provides more complex operations of 1451

Appendix

A1: MQTT Broker

A2 NCAP server

A3: Node-RED nodes

A4: Install other nodes

A5: Queueing Calculations Using Node-RED

A6: Context (global variables)

A7: MQTT on FPGA

A1: MQTT Broker

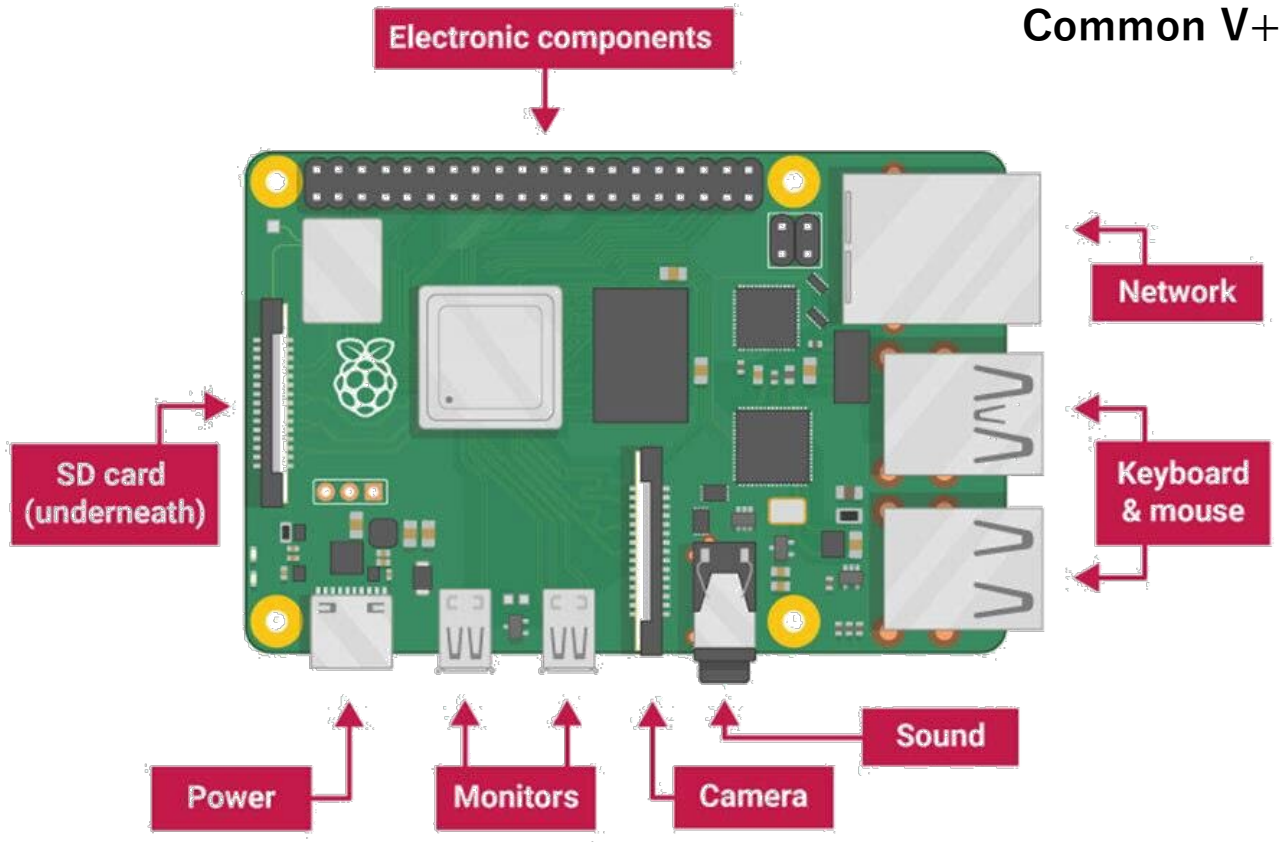
- Install Mosquitto
  - sudo apt install mosquitto
- Open mosquitto when booting
  - systemctl enable mosquitto
- Allow access from other machines
  - sudo echo “listener 1883” >> /etc/mosquitto/mosquitto.conf
  - sudo echo “allow_anonymous true” >> /etc/mosquitto/mosquitto.conf
  - sudo systemctrl restart mosquitto
- Change MQTT broker specification of Node-RED to localhost:1883
  - NCAP must send the data to the MQTT broker; Namely, it is required to install your own NCAP for publishing sensor data to the broker.

A2: NCAP server (software installation)

- Download NCAP model
  - git clone https://github.com/westlab/IEEEP1451.1.6-2023
- NCAP.py is located at IEEEP1451.1.6-2023/NCAP.py
- Install required libraries
  - pip3 install paho-mqtt PyYAML temporenc
- Edit config.yml
  - Change the server address and other settings
- Run NCAP server
  - python3 NCAP.py or ./NCAP.py

A2: NCAP server (Raspberry Pi installation)

- Raspberry-Pi 4B (3 is also Ok)
- DHT11 Temperature and Humidity Sensor
- Servo Motor (Micro Servo Motor SG-90 or compatible servo motors)



Common V+

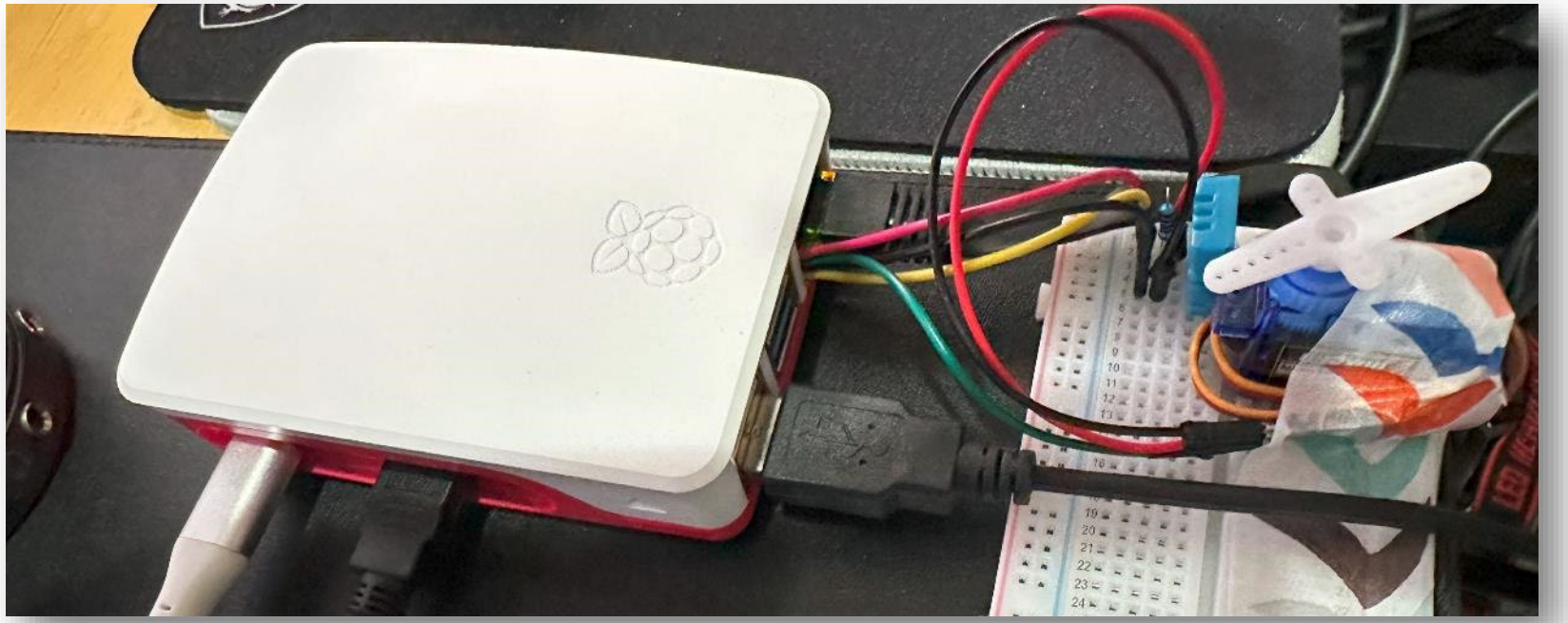
PIN	NAME	NAME	PIN
01	3.3V DC Power	5V DC Power	02
03	GPIO02 (SDA1,I²C)	5V DC Power	04
05	GPIO03 (SDL1,I²C)	Ground	06
07	GPIO04 (GPCLK0)	GPIO14 (TXD0, UART)	08
09	Ground	GPIO15 (RXD0, UART)	10
11	GPIO17	GPIO18(PWM0)	12
13	GPIO27	Ground	14
15	GPIO22	GPIO23	16
17	3.3V DC Power	GPIO24	18
19	GPIO10 (SP10_MOSI)	Ground	20
21	GPIO09 (SP10_MISO)	GPIO25	22
23	GPIO11 (SP10_CLK)	GPIO08 (SPI0_CE0_N)	24
25	Ground	GPIO07 (SPI0_CE1_N)	26
27	GPIO00 (SDA0, I²C)	GPIO07 (SCL0, I²C)	28
29	GPIO05	Ground	30
31	GPIO06	GPIO12 (PWM0)	32
33	GPIO13 (PWM1)	Ground	34
35	GPIO19	GPIO16	36
37	GPIO26	GPIO20	38
39	Ground	GPIO21	40

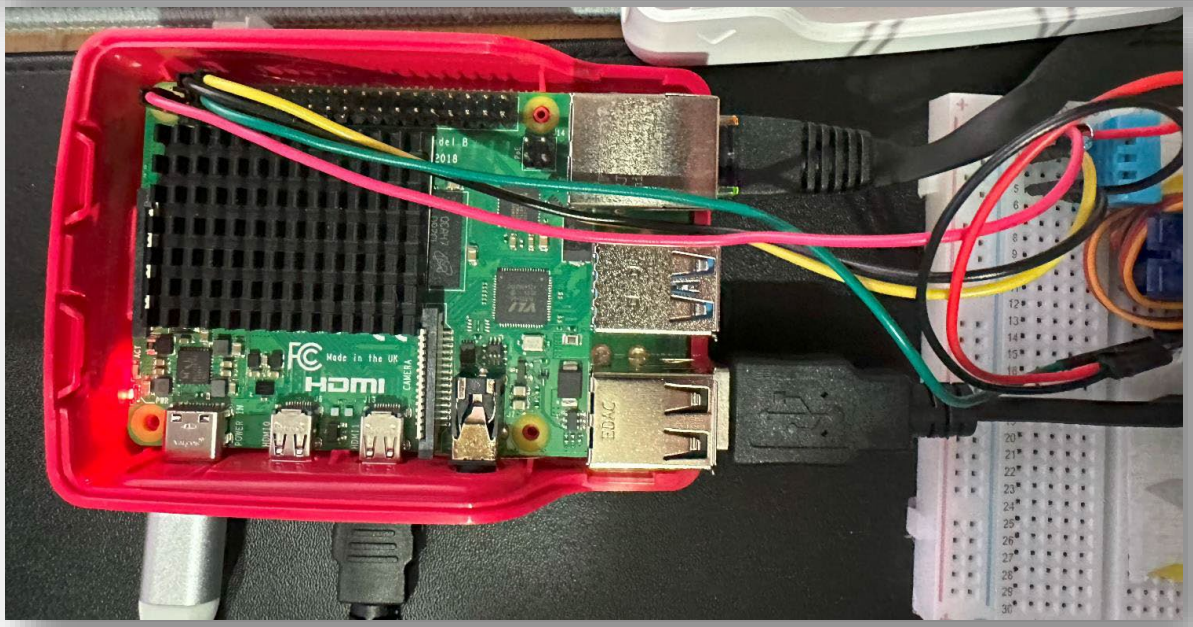
Common GND

Servo

DHT11(1K Pull Up)

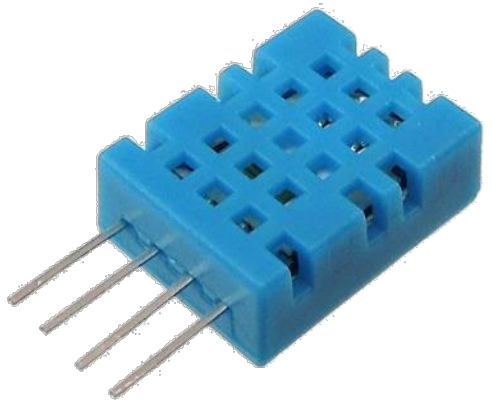
A2: NCAP/TIM implementation on my desktop





Micro Servo Motor SG-90

Temp/Humid Sensor (DTH11)



A3: Node-RED nodes (1)

- Inject nodes can be clicked in the editor to manually initiate a flow, or they can be clicked in the editor to automatically initiate a flow at regular intervals.

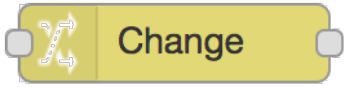
  - Outgoing messages can have payload and topic properties
    - The payload can be set to any type:
      - Flow or global context property values
      - String type, numeric type, Boolean type, buffer type, object type
      - Transmission interval (epoch milliseconds)

- Debug node displays the message in the Debug sidebar
  - Contains the time the message was received and information about the Debug node that sent the message
  - Click on the source node ID to see which node it is in the workspace
  - Enable/disable output can be controlled by buttons on the node
  - All messages can be output to the runtime log or a short (32 characters) status text can be displayed under the Debug node

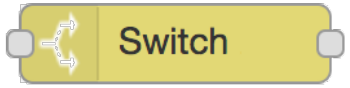
- Function node executes JavaScript code
  - For example, if you want to return the length, var newMsg = {payload: msg.payload.length}; return newMsg;

A3: Nodes in Node-RED (2)

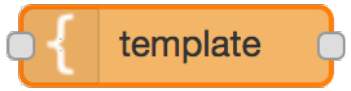
- Change node can change message properties and set context properties



  - Each node can have multiple operations applied to it in turn
    - Value Assignment - Assign properties, various types are available, existing message and context properties are also allowed
    - Value Substitution - find and replace parts of message properties
    - Move or delete values - move, rename, or delete properties; JSONata format can be set for property settings
- Switch node evaluates each message and routes messages to different flows
  - Set properties (message properties or context properties) to be evaluated
  - There are four types of rules



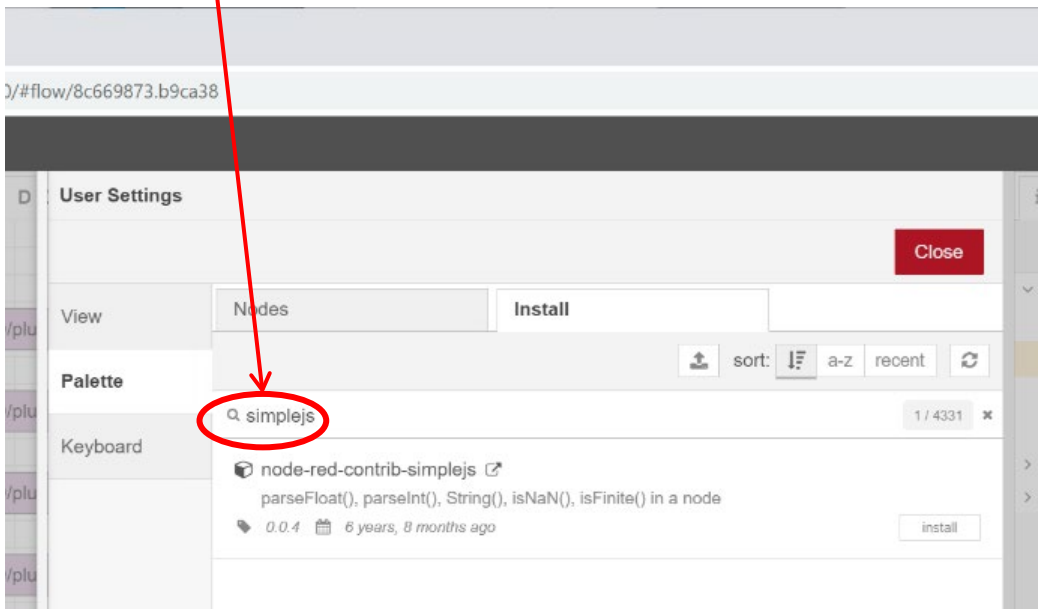
    - Value rule evaluates the set properties
    - Sequence rule can use message sequences such as those generated by Split nodes.
    - JSONata expressions evaluate the entire message and consider a match if it returns a value of true.
    - Others match if none of the aforementioned rules match.
  - It is possible to specify whether to send the message to all matching rules or to stop evaluating when a matching rule is found.
- Template node generates text from the message properties (Mustache notation).
  - For example, This is the payload: {{payload}} !

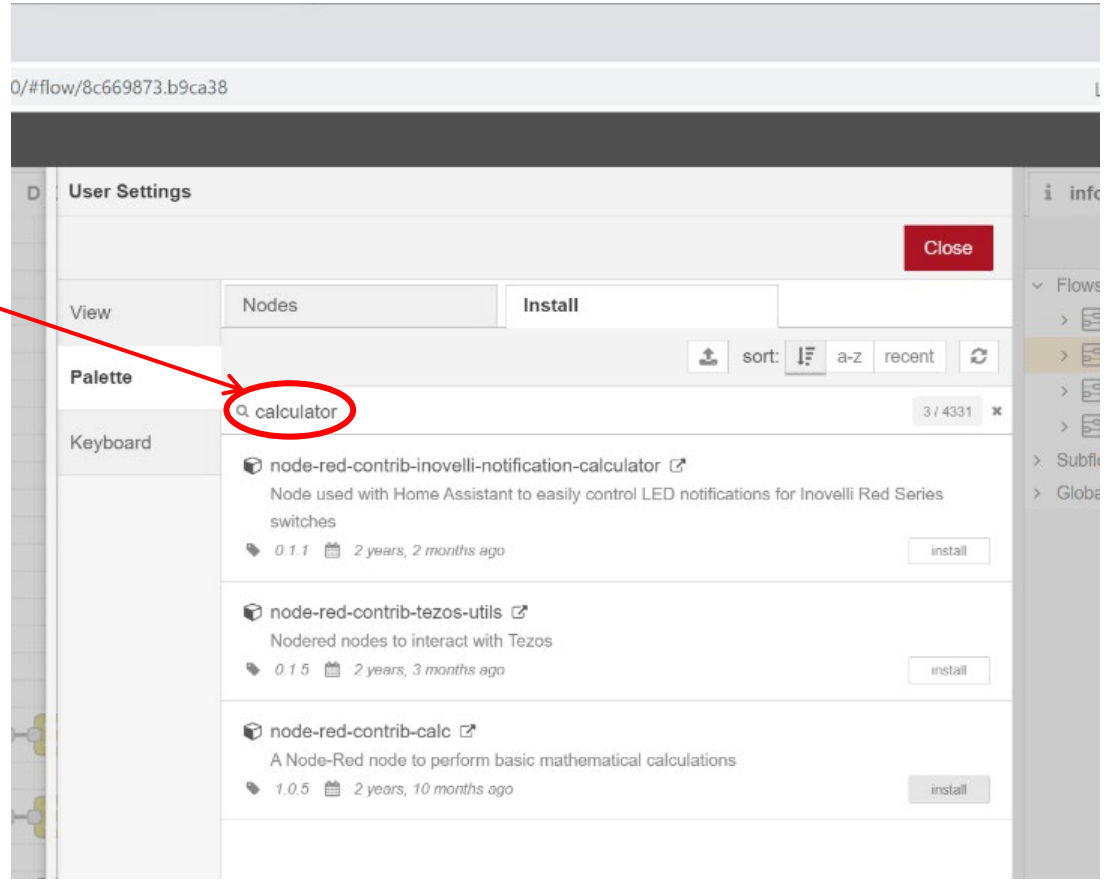


- Split node splits the flow

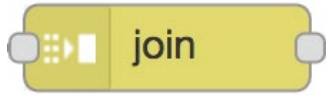
A4: Install other nodes

- There are many kinds of nodes for almost all of IoT and Cloud services, such as ChatGTP.
- Useful examples
  - Install calculator for calculations
  - Install simplejs for parseFloat(), parseInt(), String(), isNaN(), isFinite() in a node

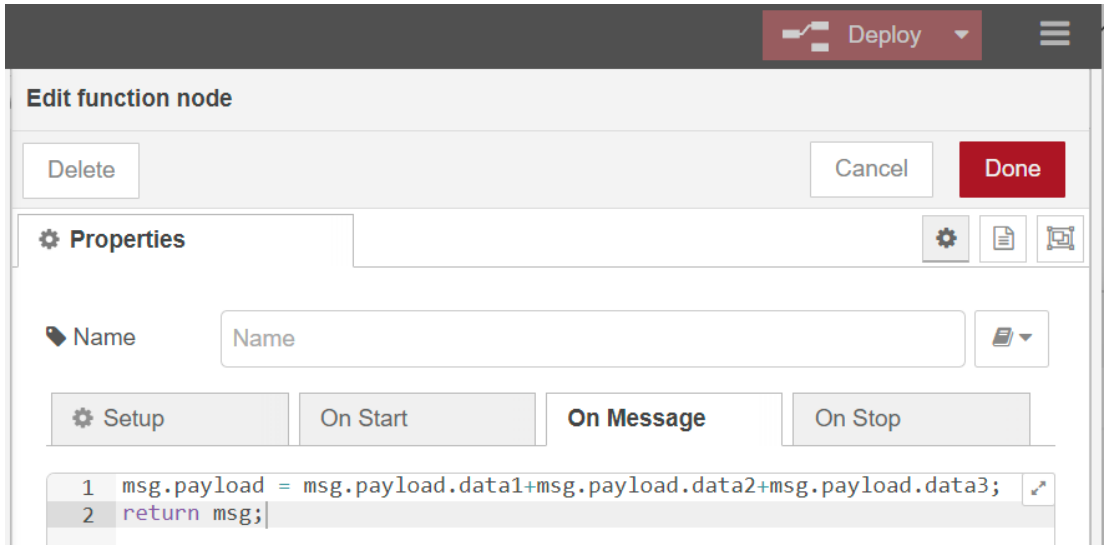


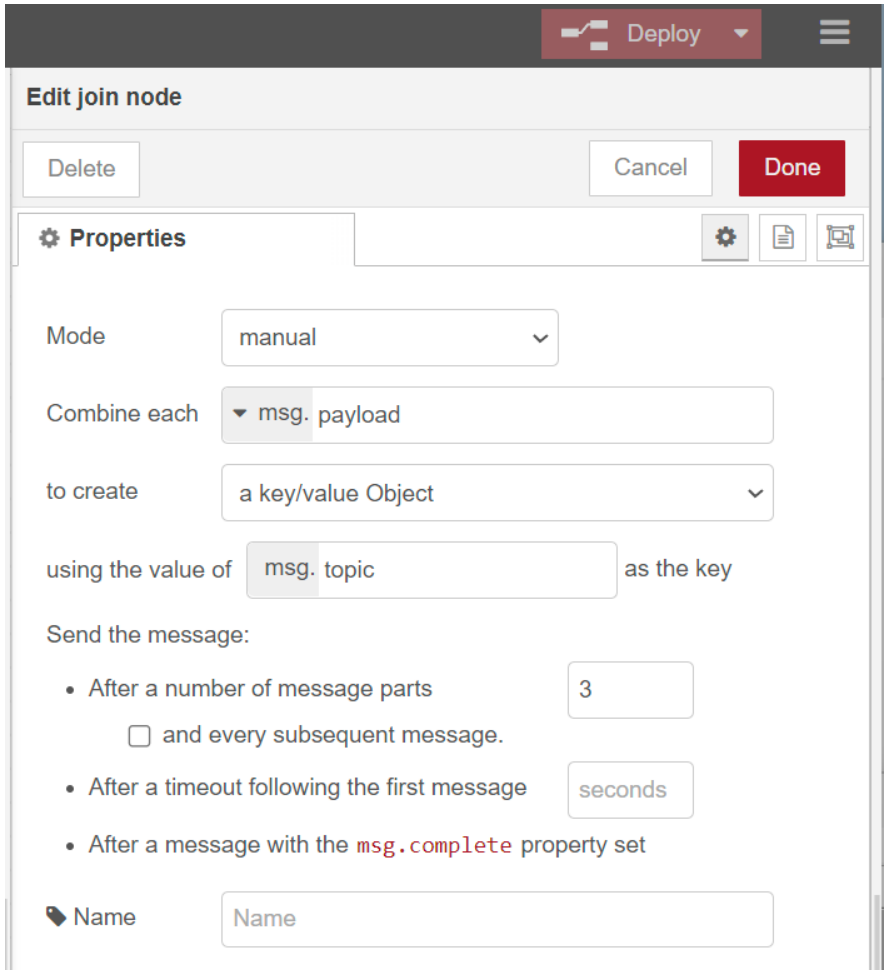


A5: Queueing Calculations Using Node-RED



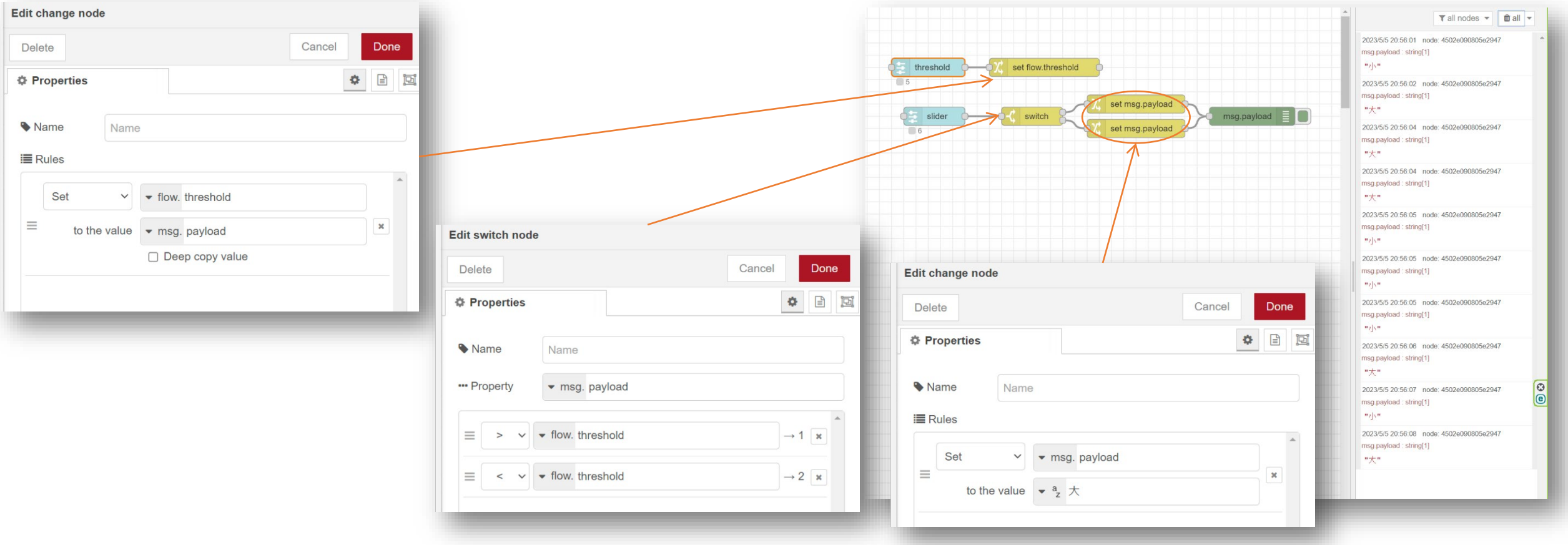
- Node-RED is a bit cumbersome to "calculate between multiple streams".
  - Of course, since they are different streams, the values do not come at the same time
- Node-RED uses a clever way to do this
  - Name the message by attaching a topic to the message
  - Using a join node, configure as follows
  - Specify the number of streams to mix in
    "after receiving the specified number of message parts".
  - Finally, add in function
  - The name becomes msg.payload.topic





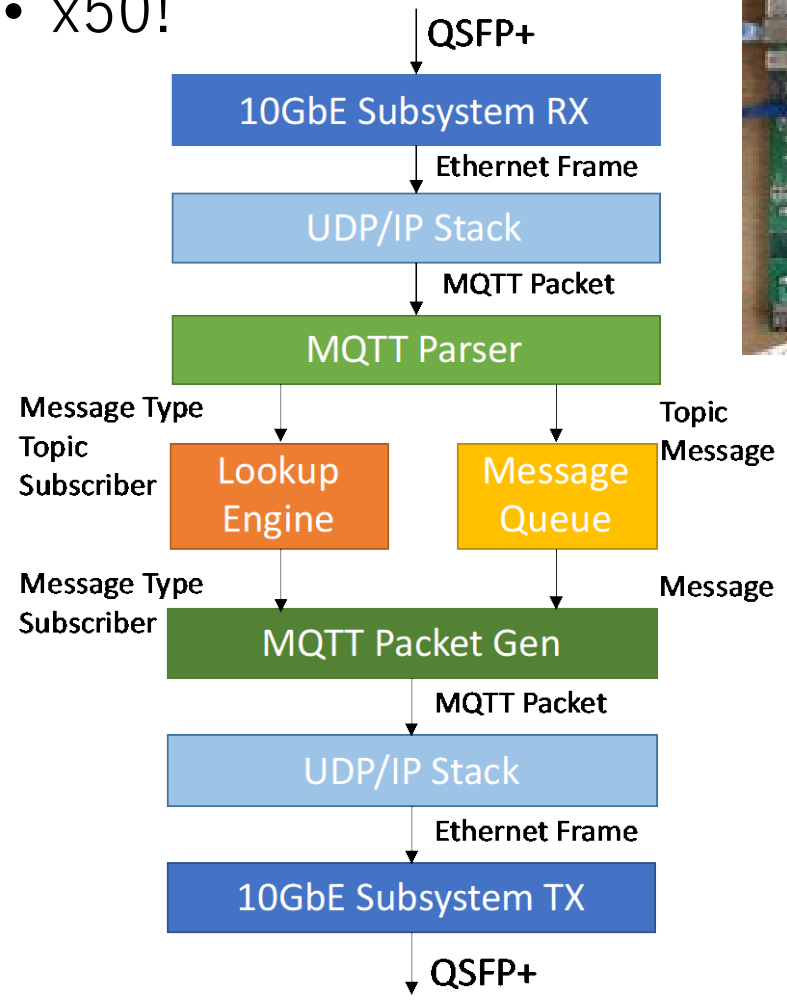
A6: Context (global variables)

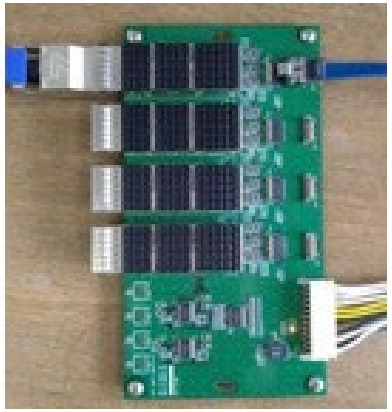
- Node-RED has contexts that are equivalent to global variables.
  - You can assign or reference values to contexts.
- Example: Design a system with two sliders of threshold and input. Output “BIG" if the input value is greater than the threshold slider and “SMALL“, vice versa.

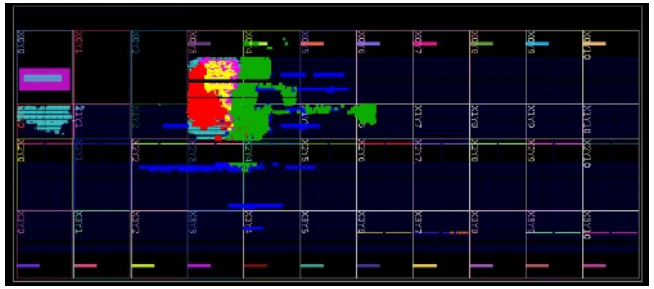


A7: MQTT on FPGA

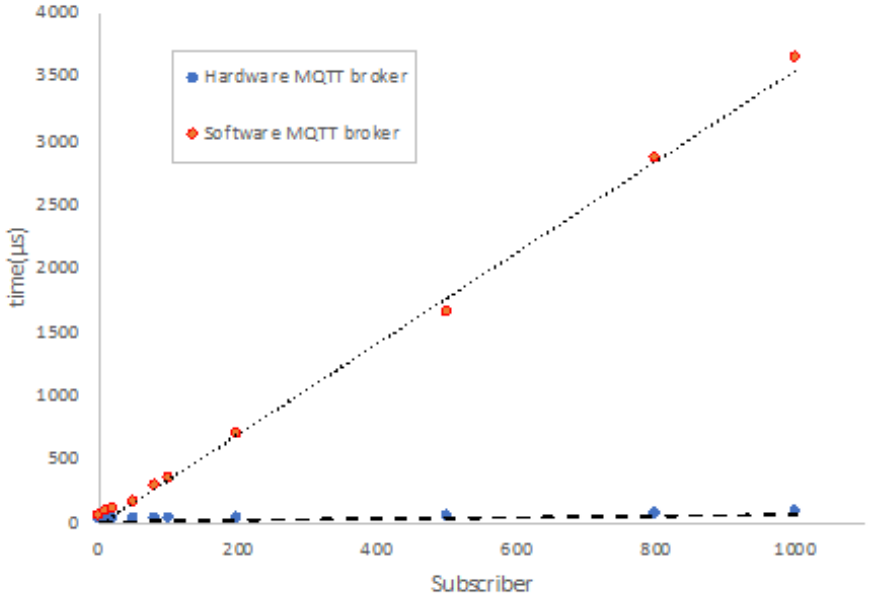
- MQTT accelerator as a hardware device implemented on FPGA
- x50!



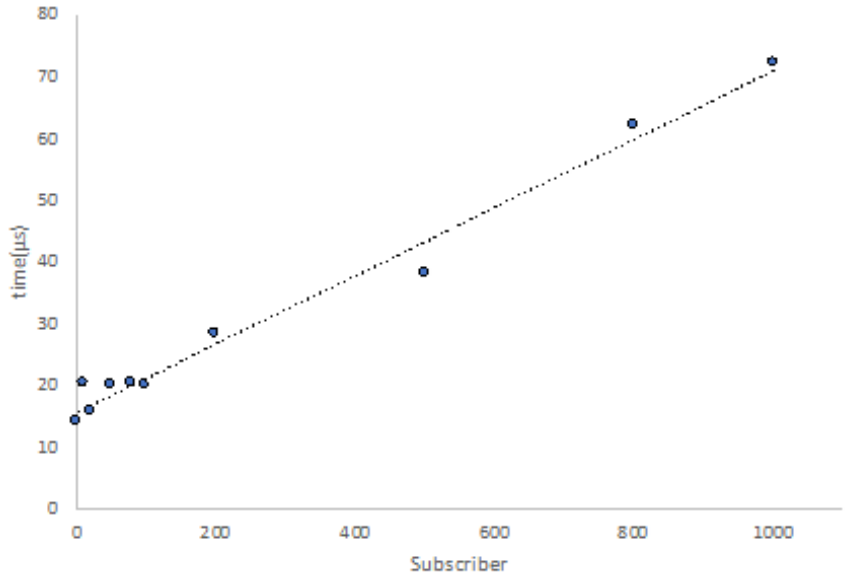




Module	FF	LUT	BRAM
MQTT Parser	1,015	3,343	0
Lookup Engine	265	501	108
Message Queue	24	1,652	18.5
MQTT Packet Generator	1,938	296	0
UDP/IP Stack & 10GbE Subsystem	7,550	6,479	6
Total	21,901	18,532	151
Available	1,045,440	522,720	984



(a) software-implemented broker



(b) hardware-implemented broker