acontis technologies GmbH

SOFTWARE

EC-Monitor

Quick Start Guide

Version 3.2

Edition: August 1, 2025

Contents

1 About **4**

2 Get Your License Key **5**

3 Install EC-Monitor **6**
3.1 Prepare the Operating System (Windows) . . . 6
3.2 Prepare the Operating System (Linux) . . . 6

4 Architecture **8**

5 EtherCAT® Network Information (ENI) **9**

6 Running the EC-Monitor Demo **10**
6.1 Data Acquisition . . . 10
6.2 Extending the EC-Monitor Demo . . . 12

7 Running the EC-Monitor MQTT Demo **14**
7.1 Setting up Eclipse Mosquitto MQTT broker . . . 14
7.2 Processing data with Node-RED . . . 15
7.3 Visualizing data with Node-RED . . . 18

1 About

The EC-Monitor Software Development Kit (SDK) offers the possibility for Data Tracing / Listening / Sniffing / Logging Diagnosis and Monitoring of EtherCAT® Networks. It's suitable for new (Greenfield) and existing (Brownfield) installations. Also it's independent from EtherCAT® MainDevice Controller Software and Hardware. A TAP device is required to observe the network traffic at a specific location in an EtherCAT® network.

This Quick Start Guide briefly shows how to install and run the included example program. The EC-Monitor's User Manual contains information that is more detailed and is available at https://developer.acontis.com/ec-monitor.html.

The EC-Monitor can be obtained from https://www.acontis.com/en/ecdownloads.html.

If you have questions, please contact us at https://www.acontis.com/en/contactform.html.

2 Get Your License Key

A valid license key is needed to run EC-Monitor, see chapter "*Protected version*" in the EC-Monitor User Manual for how to obtain it.

It is possible to continue this Quick Start Guide without a valid license key with limited functionality.

The license key **must match the network adapter** used by EC-Monitor!

3 Install EC-Monitor

The EC-Monitor SDK including its examples is contained in the installation package named e.g. `EC-Monitor-V3.2.2.05-Windows-x86_64Bit-Eval.zip`. The procedure is generally the same for all operating systems supported by the EC-Monitor.

3.1 Prepare the Operating System (Windows)

The installation of EC-Monitor for Windows contains the installer `setup.msi` guiding through the installation process. The acontis ECAT Protocol Driver is needed to use the Ethernet Driver NDIS. It can be installed from

- `<InstallPath>\Bin\Windows\x64\EcatNdisSetup-x86_64Bit.msi`
- `<InstallPath>\Bin\Windows\x86\EcatNdisSetup-x86_32Bit.msi`

The command “ipconfig” shows the IP settings of the installed network adapters, including the IP address of the dedicated Ethernet network adapter for EC-Monitor e.g., “192.168.99.1”.

3.2 Prepare the Operating System (Linux)

For non-Windows operating systems, the files of the SDK to be extracted are directly contained within the installation package. It is recommended to copy the files to a folder where the user has write access.

EC-Monitor supports different Ethernet adapter types.

It is recommended to use an Intel Pro/1000 handled by the acontis emllIntelGbe driver. Other network adapter types are supported too and described in the EC-Monitor User Manual.

1. The acontis atemsys Linux Kernel driver must be downloaded and applied to the Linux system: See https://github.com/acontis/atemsys.
2. The commands “lshw -short -c network” and “lspci | grep Ethernet” show the hardware information of the installed network adapters, including the PCI bus address of the dedicated Ethernet network adapter for EC-Monitor, e.g. “01:00.0” (bus 01, device 00, function 0 as needed below):

   ```
   $ sudo bash
   $ lshw -short -c network
   $
   $ H/W path        Device      Class          Description
   $ ============================================================
   $ /0/100/1/0      enp1s0f0    network        I350 Gigabit Network Connection
   $ /0/100/19       lan         network        Ethernet Connection I217-LM
   $
   $ lspci | grep Ethernet
   $ 00:19.0 Ethernet controller: Intel Corporation Ethernet Connection I217-LM
   $ 01:00.0 Ethernet controller: I350 Gigabit Network Connection
   ```

   The PCI bus address is used to specify the network adapter adapter used by EC-Monitor Demo and is formatted as 0x01bbddff:

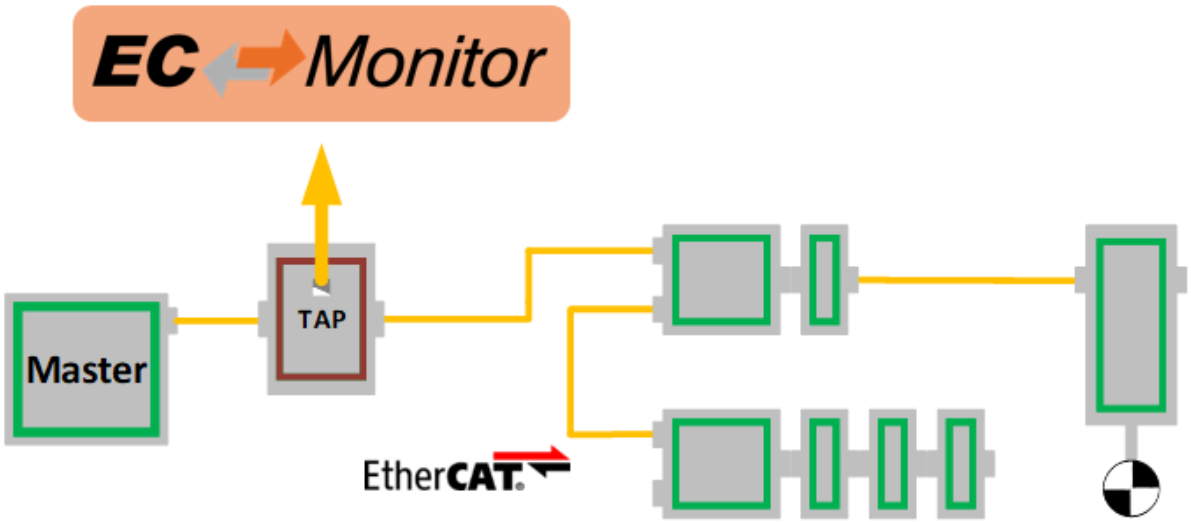
   - *bb* Bus Number
   - *dd* Device Number
   - *ff* Function Number
3. The dedicated network adapter to be used must be unbound from the Linux Kernel and atemsys loaded:

   ```
   $ echo "0000:01:00.0" > /sys/bus/pci/drivers/igb/unbind
   $ modprobe atemsys
   ```

Unbinding the network adapter instance from the Linux Kernel and loading the atemsys Kernel driver is non-persistent and **must be redone after reboot.**

4 Architecture

The Ethernet TAP device should be inserted in the network between the EtherCAT® MainDevice and the first SubDevice or, if this is not possible, between two SubDevices. The position of the TAP is detected automatically and the EtherCAT® traffic is processed accordingly.



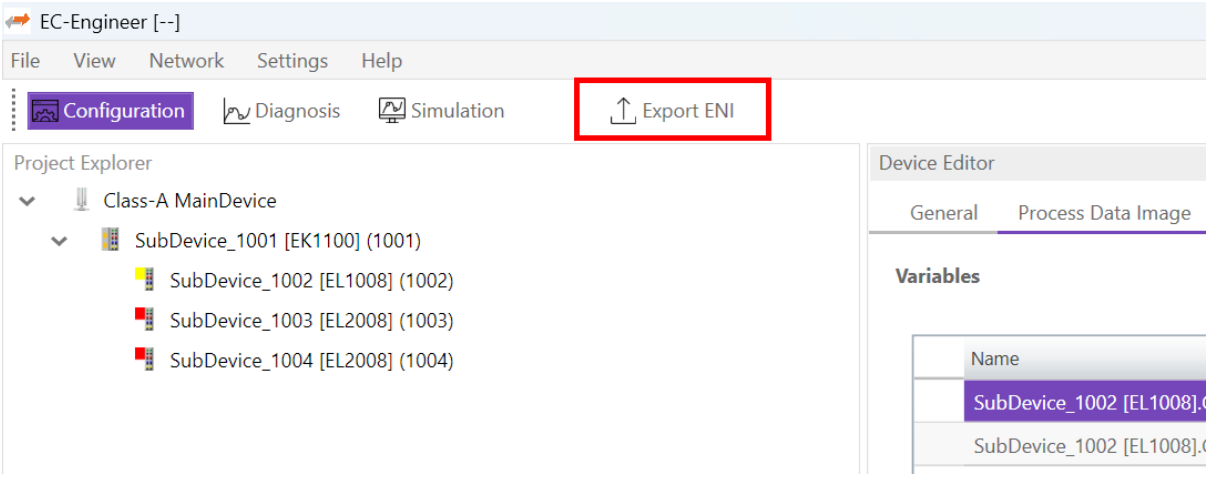
Different positions of the TAP device are possible. For more information about available options and their limitations, see chapter *"TAP positioning"* in the EC-Monitor User Manual.

5 EtherCAT® Network Information (ENI)

The EC-Monitor needs knowledge about the network to be observed which must be configured using a configuration file in the EtherCAT® Network Information (ENI) format. It is strongly recommended to use the same configuration file for MainDevice and EC-Monitor.

The EtherCAT® Network can be configured using EC-Engineer, which can be obtained from https://www.acontis.com/en/ecdownloads.html.

See the EC-Engineer User Manual or tutorial videos at https://developer.acontis.com/ec-engineer. After configuring the network, press **`Export ENI`** to export the ENI file.



6 Running the EC-Monitor Demo

After the EC-Monitor is installed and the network adapter connected to the TAP device and the EtherCAT® Network Information (ENI) file are prepared, the demo can be run.

- Connect the network as specified by the ENI file. Insert the TAP device, ideally between the MainDevice and the first SubDevice.
- Open a terminal to `<InstallPath>\Bin\Windows\x64`.
- On Windows, run

```
$ ./EcMonitorDemo -ndis <adapter ip address> 1 -f <path to eni.xml> -rec <path
↪  to record>
```

- On Linux, run

```
$ ./EcMonitorDemo -sockraw <network link name> 1 -f <path to eni.xml> -rec
↪  <path to record>
```

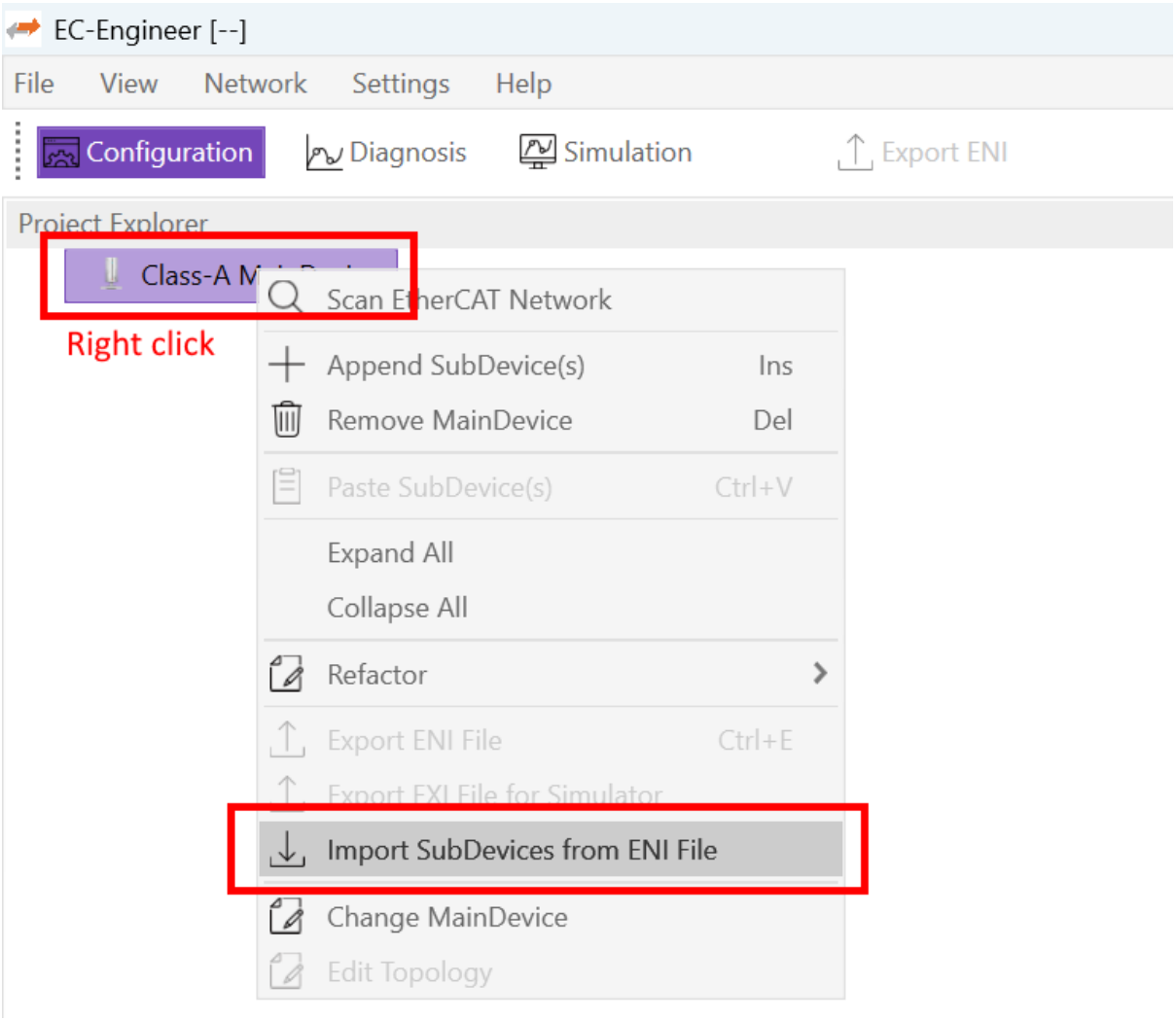
The respective adapter ip address or network link name connected to the TAP device has to be used, see *Install EC-Monitor*. For other operating systems, see chapter *"Platform and Operating Systems (OS)"* in the EC-Monitor User Manual. Alternatively, the **`-play <path to *.pcap / *.pcapng>`** parameter can be used to process previously recorded instead of live traffic.

The EC-Monitor will record the EtherCAT® network traffic in pcap-ng format to the file provided to the **`-rec`** parameter. For a full list of command line parameters, see chapter *"Command line parameters"* in the EC-Monitor User Manual.

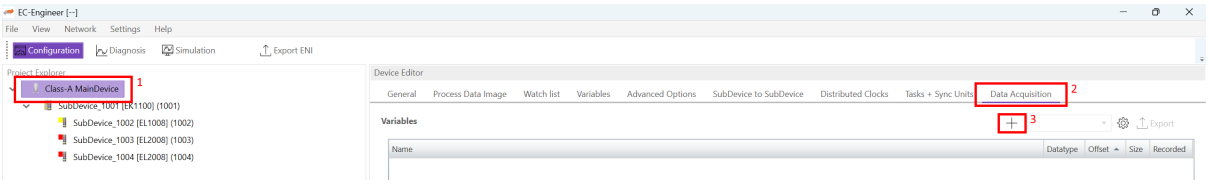
6.1 Data Acquisition

The EC-Monitor Demo is capable of periodically recording variable values of SubDevices in the EtherCAT® network in MF4 and CSV formats.

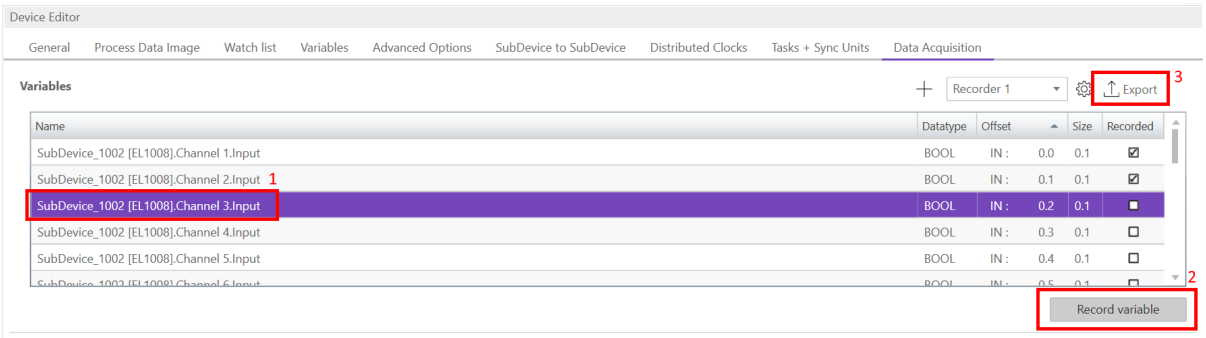
In EC-Engineer, configure the network or import the EtherCAT® Network Information (ENI) file.



After the network is configured, select the MainDevice, navigate to the **`Data Acquisition`** tab, and press **`+`** to create a new record file. Select the file format (MF4 or CSV) and path as needed.



Select the variable(s) to be included in the record and press **`Record variable`** for each. After all desired variables have been selected, press **`Export`** to export the recorder configuration file to the desired path.



Finally, run the EC-Monitor Demo according to *Running the EC-Monitor Demo*, but replace the **`-rec <path to record>`** parameter with **`-daqrec <path to recorder.xml>`**. The output file according to the recorder configuration file will contain the recorded data.

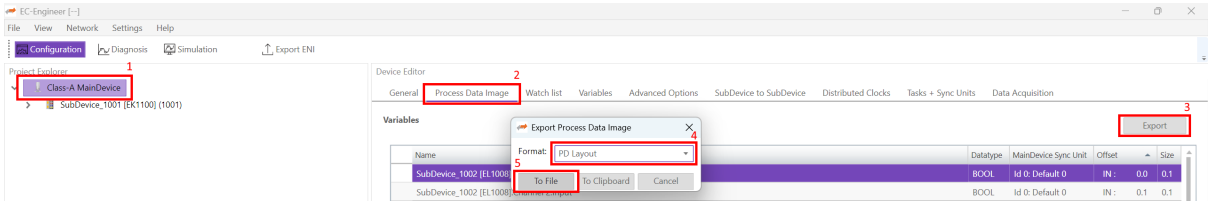
6.2 Extending the EC-Monitor Demo

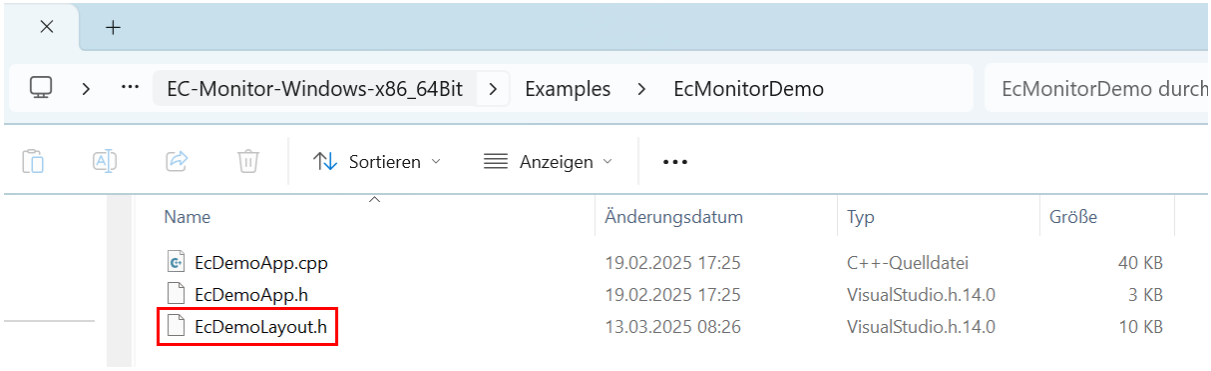
The source code of the EC-Monitor Demo is included with each installation. It is located in the `<InstallPath>\Examples` folder. The EC-Monitor Demo serves as an application template and can be expanded as needed, see chapter *"Software Integration"* of the EC-Monitor User Manual.

The EC-Monitor SDK provides direct access to the process data of the EtherCAT® network, see chapter *"Process Data Access"* of the EC-Monitor User Manual. The demo will be extended to print a notice to the command line each time a specific SubDevice variable changes.

In EC-Engineer, configure the network or import the EtherCAT® Network Information (ENI) file. See *Data Acquisition* for a screenshot how to import the file.

After the network is configured, select the MainDevice, navigate to the **`Process Data Image`** tab, and press **`Export`** to export the PDI layout. Select the **`PD Layout`** format and press **`To File`** to generate a header file for the extension. Save the file to `<InstallPath>\Examples\EcMonitorDemo\EcDemoLayout.h`.





Open the project in Visual Studio. The project file can be found in `<InstallPath>\Workspace\WindowsVS2015\EcMonitorDemo\EcMonitorDemo.vcxproj`. Add the newly created header file to the project.

The file `EcDemoApp.cpp` is designed to be extended with custom functionality. Include the new header file here.

```
#include "EcDemoLayout.h"
```

First, add a member to the type `_T_MY_APP_DESC` to hold the status of the variable. We will observe a 1 bit variable, so the type `EC_T_BYTE` is appropriate.

```
typedef struct _T_MY_APP_DESC
{
    // ...
    // Add the following line
    EC_T_BYTE       bSub1002In1Status;
} T_MY_APP_DESC;
```

The function **myAppWorkPd** is called for each observed cycle, thus it is appropriate to add our custom processing logic here.

```
static EC_T_DWORD myAppWorkpd(T_EC_DEMO_APP_CONTEXT* pAppContext)
{
    // ...
    // We are processing an input variable, so we get the PDI input pointer
    EC_T_BYTE* pbyPdIn = emonGetProcessImageInputPtr(pAppContext->dwInstanceId);

    // The type and offset of the variable are defined in the exported header file
    // We add the offset to the PDI input pointer, cast it to the target type, and␣
↪dereference our desired member
    EC_T_BYTE currentStatus = ((_T_PDLAYOUT_IN_SUBDEVICE_1002*)(pbyPdIn + PDLAYOUT_
↪IN_OFFSET_SUBDEVICE_1002))->byChannel_1_Input;

    // Finally, we print to the console when the variable has changed
    if (pMyAppDesc->bSub1002In1Status != currentStatus) {
        pMyAppDesc->bSub1002In1Status = currentStatus;
        printf("Variable has changed: %u\n", currentStatus);
    }

    return EC_E_NOERROR;
}
```

Compile the application. Copy the necessary dll files from `<InstallPath\Bin\Windows\x64>` to `<InstallPath\Bin\Windows\x64\Debug>`.

- EcDaq.dll
- EcMonitor.dll
- EcMonitorRasServer.dll
- emllNdis.dll, if NDIS Ethernet Driver will be used

Finally, run the application according to *Running the EC-Monitor Demo*. The **-rec <path to record>** parameter can be omitted. Once the EtherCAT® network is in operational state, the console output is visible whenever the designated SubDevice variable changes.

7 Running the EC-Monitor MQTT Demo

MQTT (https://docs.oasis-open.org/mqtt/mqtt/) is a lightweight publish/subscribe messaging protocol using a message broker.

In addition to the regular demo, the EC-Monitor includes a MQTT demo. This client regularly publishes all input and output variables of all configured SubDevices to a MQTT broker. Each variable is published under a unique topic. The payload is a byte buffer including various information. See section *Running EcMonitorDemoMqtt* of the EC-Monitor User Manual for dependency setup and details about the topics and the buffer layout.

Variables can be published cyclically as well as based on change detection in the process image, both with a configurable minimum time between messages.

Run the EcMonitorDemoMqtt, similarly to *Running the EC-Monitor Demo*. Add a parameter to connect to the MQTT broker.

`-mqtt <broker address>`

Additionally, the minimum times between messages can be configured in milliseconds:

`-mqttCycTime <time ms>` for cyclic messages

`-mqttChgTime <time ms>` for messages based on change detection

It is recommended to set at least one of the time parameters. If neither is set, all variables are published on every cycle, which can exhaust the internal message queue.

7.1 Setting up Eclipse Mosquitto MQTT broker

If no MQTT broker is available yet, the Eclipse Mosquitto broker can be set up quickly using Docker.

1. Create the Mosquitto configuration file:

   `<workingDirectory>/mosquitto/config/mosquitto.conf`.

2. Add configuration for the listener port and anonymous authentication. Optionally, enable verbose logging to later see if the EC-Monitor MQTT Demo can connect and publish messages successfully:

   ```
   listener 1883
   allow_anonymous true
   log_type all
   ```

3. Launch the container using the following command:

   ```
   $ docker run -it -p 1883:1883 -v "$PWD/mosquitto/config:/mosquitto/config"
   ↪  eclipse-mosquitto
   ```

4. Run the EcMonitorDemoMqtt, similarly to *Running the EC-Monitor Demo*, and connect to the broker using the parameter **`-mqtt localhost`**

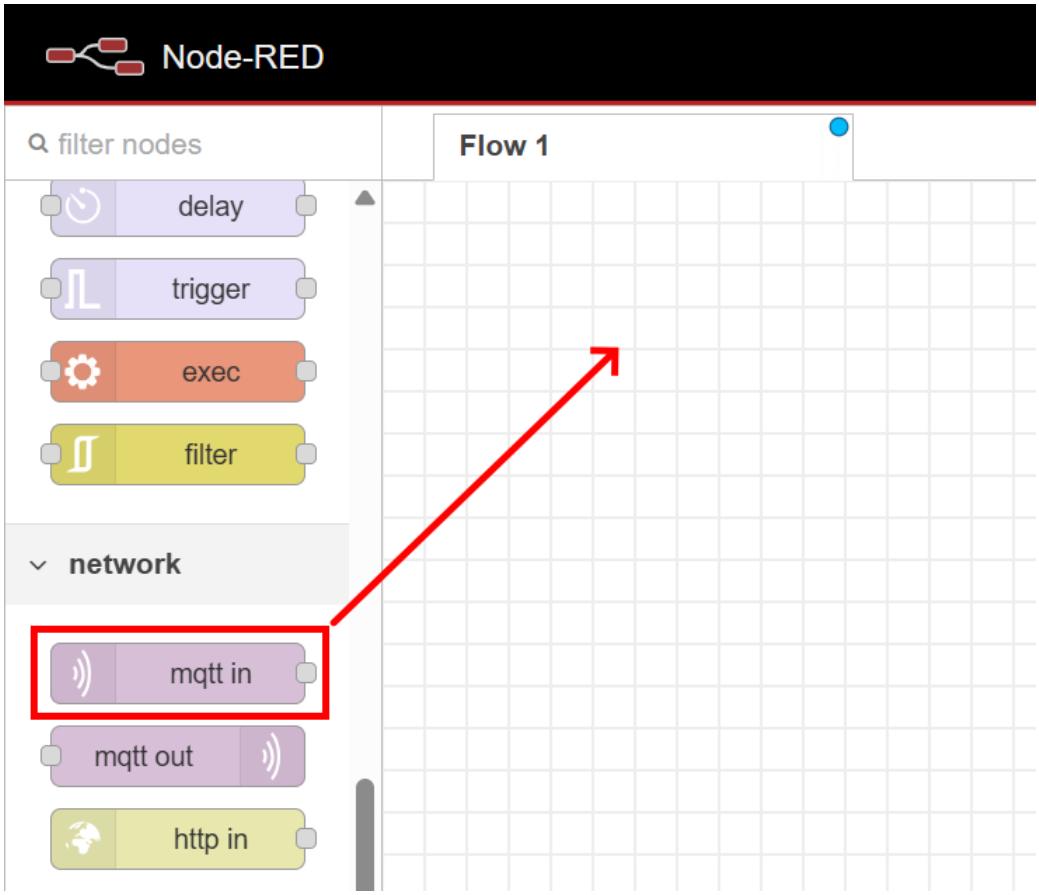
The container output will show a notice for each message published by the EC-Monitor MQTT Demo, e.g.

```
1744109196: Received PUBLISH from EcMonitor0 (d0, q0, r0, m0, '/EtherCAT/Monitor0/
↪slavebyname/SubDevice_1006 [EC-Training Generator]/variable/input/SubDevice_1006␣
↪[EC-Training Generator].Counter1.NetworkClock', ... (14 bytes))
```

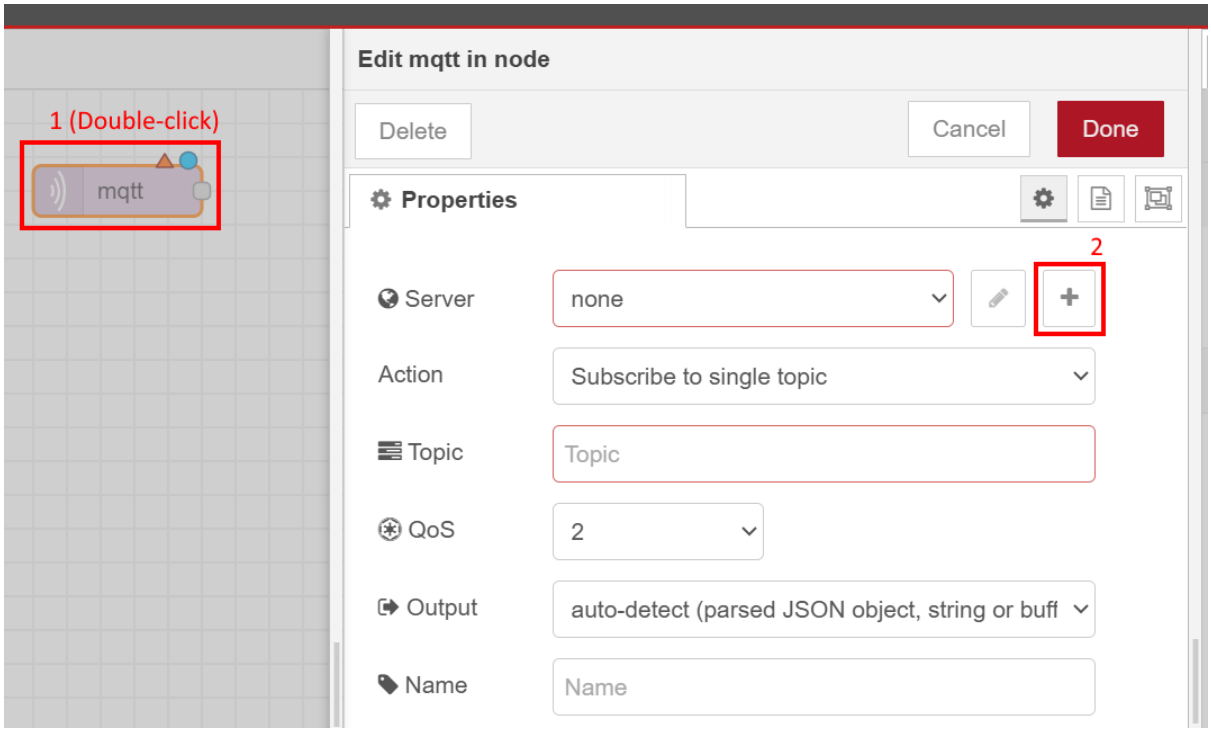
7.2 Processing data with Node-RED

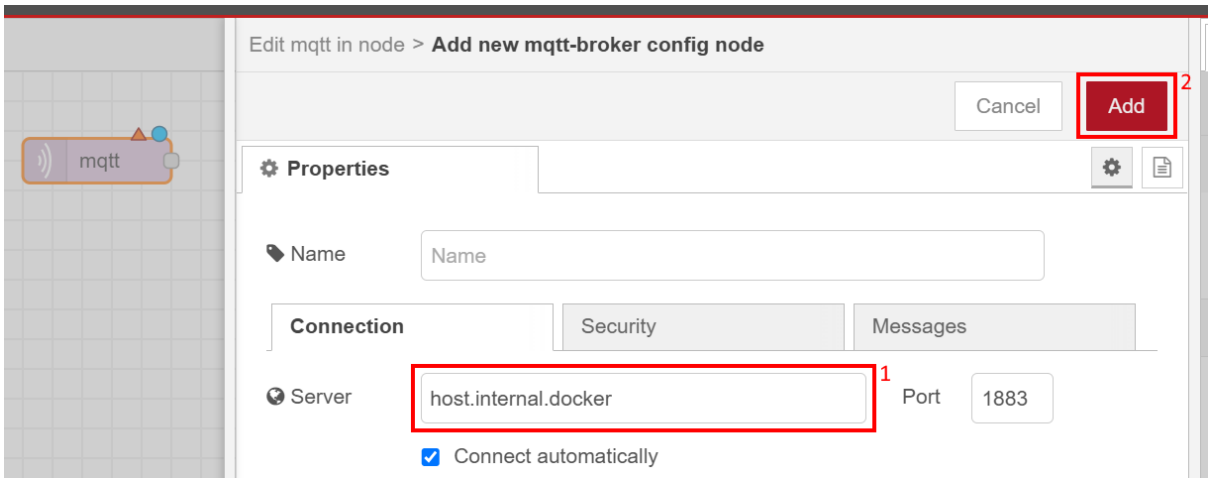
Node-RED (https://nodered.org) is a low-code programming environment for data processing. See the Node-RED documentation for setup instructions and open the web-based configurator.

1. The *"MQTT in"* node can be used to connect to a MQTT broker and subscribe to a topic. Drag and drop it to the flow.

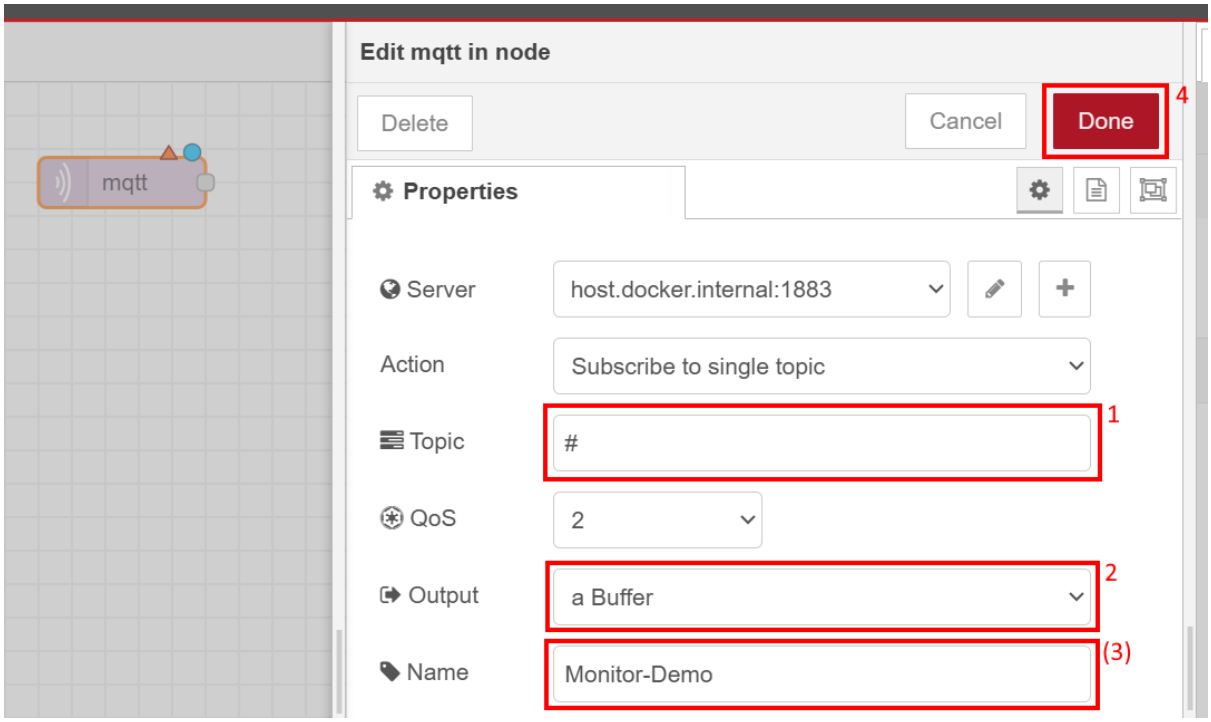


2. Double click the new node to open the configuration. If a MQTT broker is already configured, select it from the server dropdown, otherwise press the **+** sign next to it. Enter the broker address in the server field. Press **Add** to save the MQTT broker connection. If Node-RED is running in a Docker container on Windows, **`host.internal.docker`** can be used to address the host.

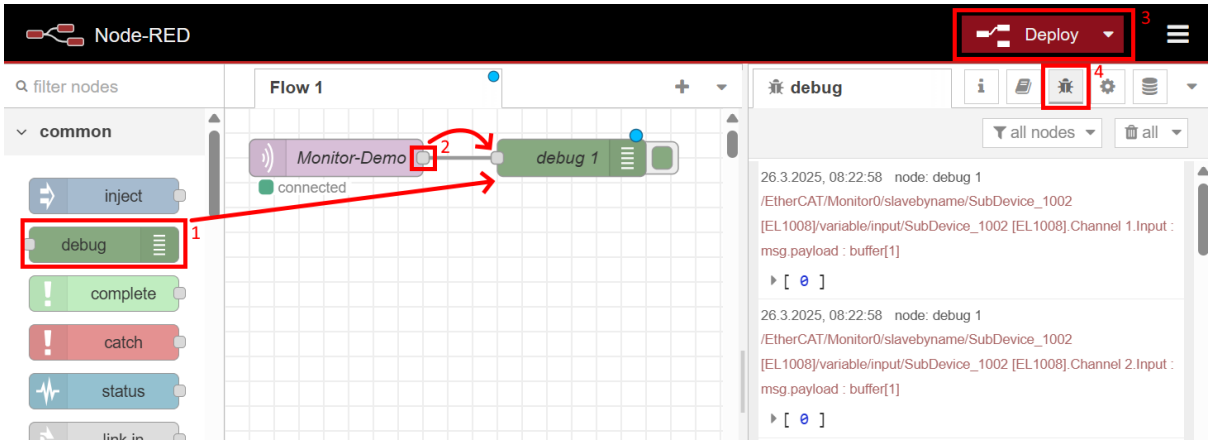




3. Set the topic to **#**, the wildcard topic which matches all topics, see the MQTT specification (https://docs.oasis-open.org/mqtt/mqtt/). Choose the output type **`buffer`**. Press **`Done`** to save the node configuration.

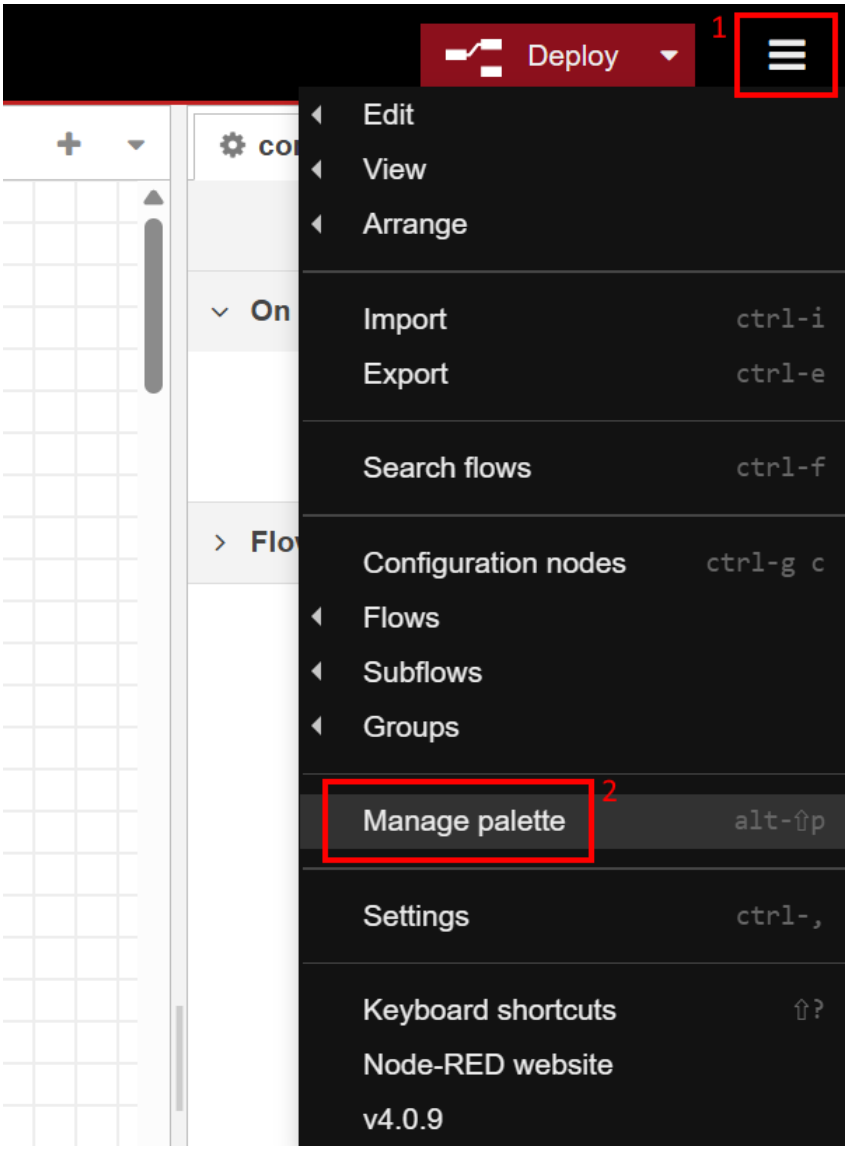


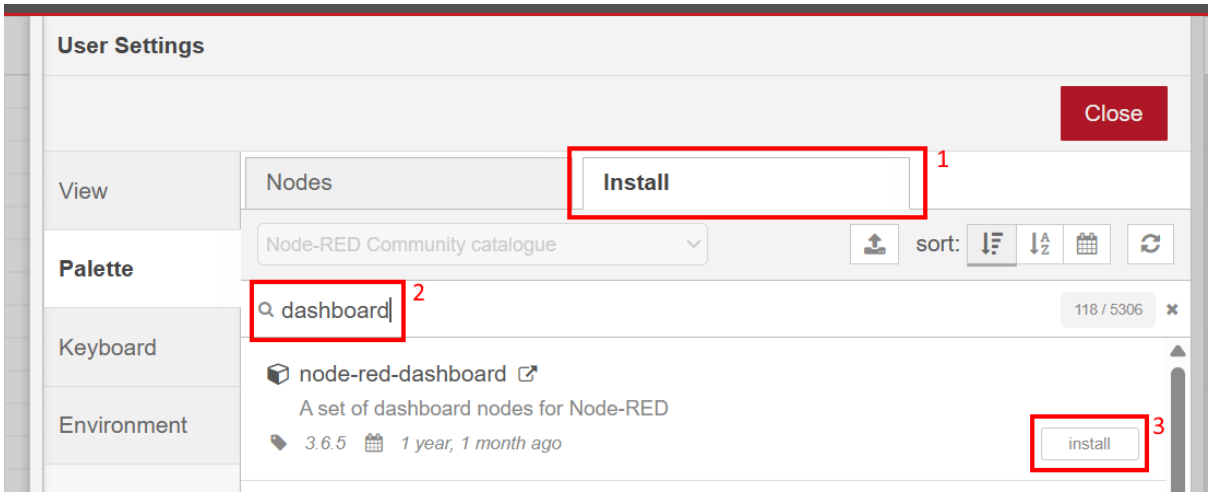
4. Add and connect a debug node to view the output data of the MQTT in node. Press **`Deploy`** to launch the flow on the Node-RED instance. If everything is configured correctly, the MQTT in node reads *"Connected"*.
5. Press the bug icon to open the debug view. Run the EC-Monitor MQTT Demo according to *Running the EC-Monitor MQTT Demo* and the variable data begins to appear in the debug view.



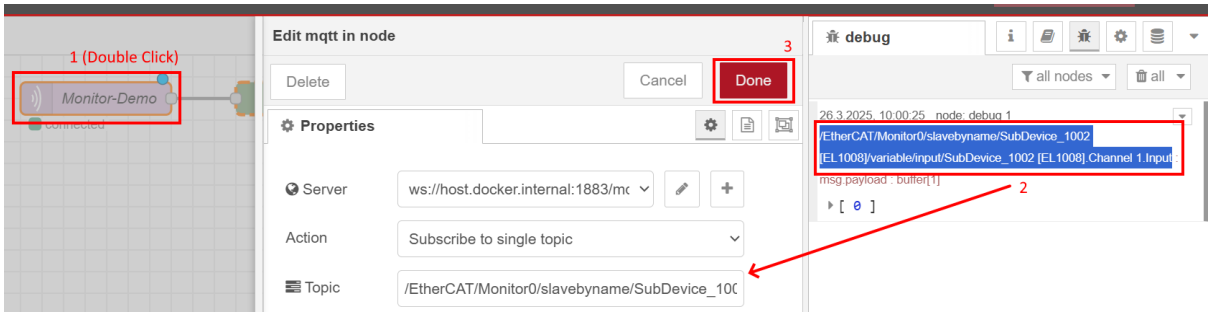
7.3 Visualizing data with Node-RED

Node-RED can be extended with many more nodes than it ships by default. Go ahead and install the **`node-red-dashboard`** module via the palette manager.

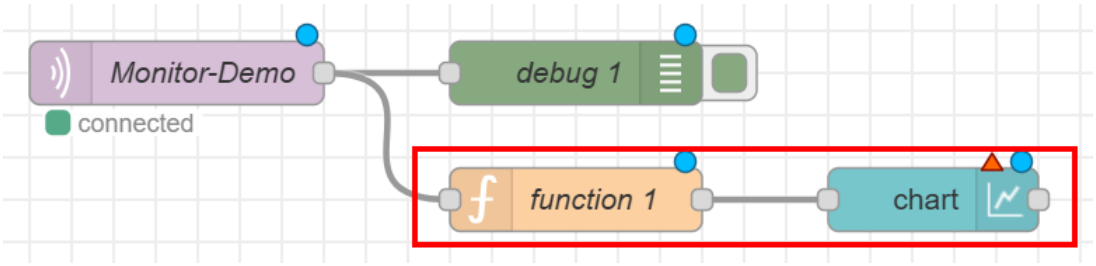




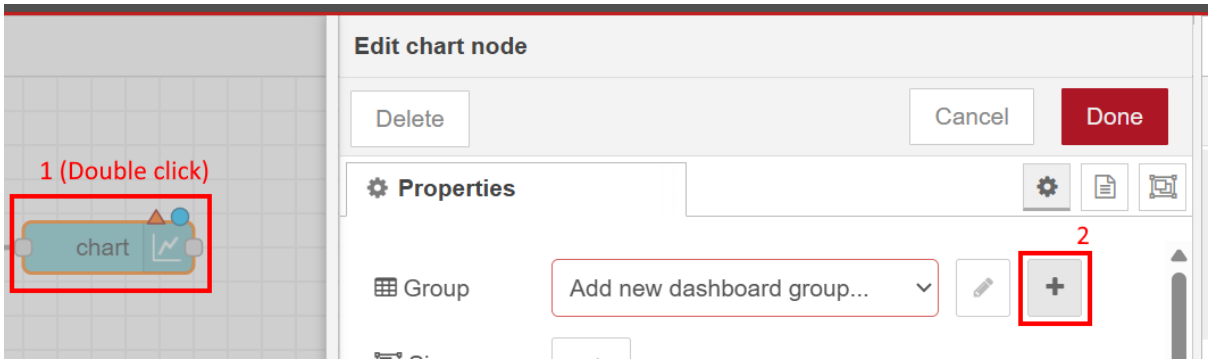
Double click the MQTT in node to open the configuration. Copy the topic of the variable to be visualized to the topic property of the node and press **Done** to save the changes.



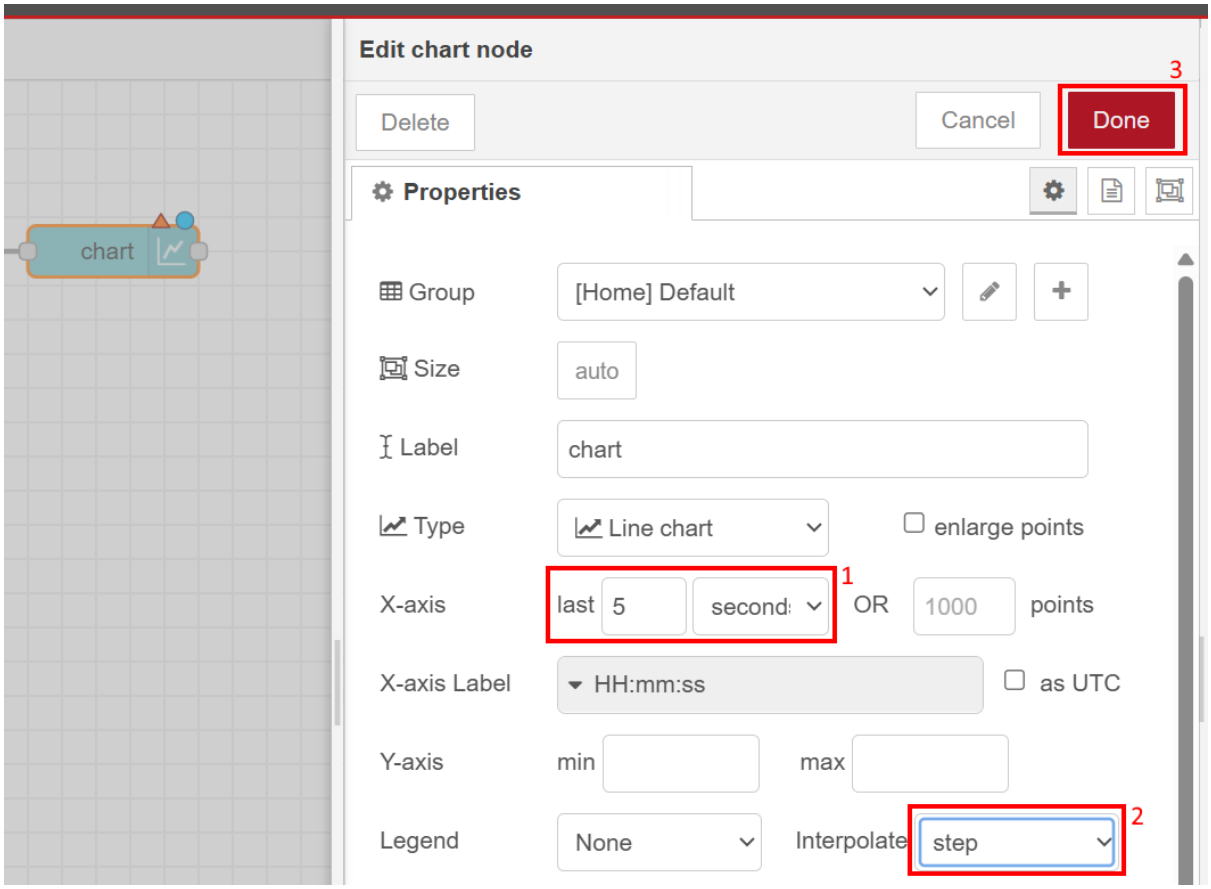
If the flow is now **Deployed** and demo launched, the debug view only shows messages for the desired variable. Add a function and a chart node to the flow and connect them to the MQTT in node in this order.



Double click the chart node to configure it. If a dashboard group is already configured, select it from the group dropdown, otherwise press the **+** sign next to it. Proceed accordingly for the dashboard tab within the dashboard group configuration. Press **Add** until the chart configuration is visible again.



Choose an appropriate timespan and interpolation for the variable. Because we are using a digital input, we will choose to visualize the last 5 seconds with step interpolation. Press **Done** to save the changes.

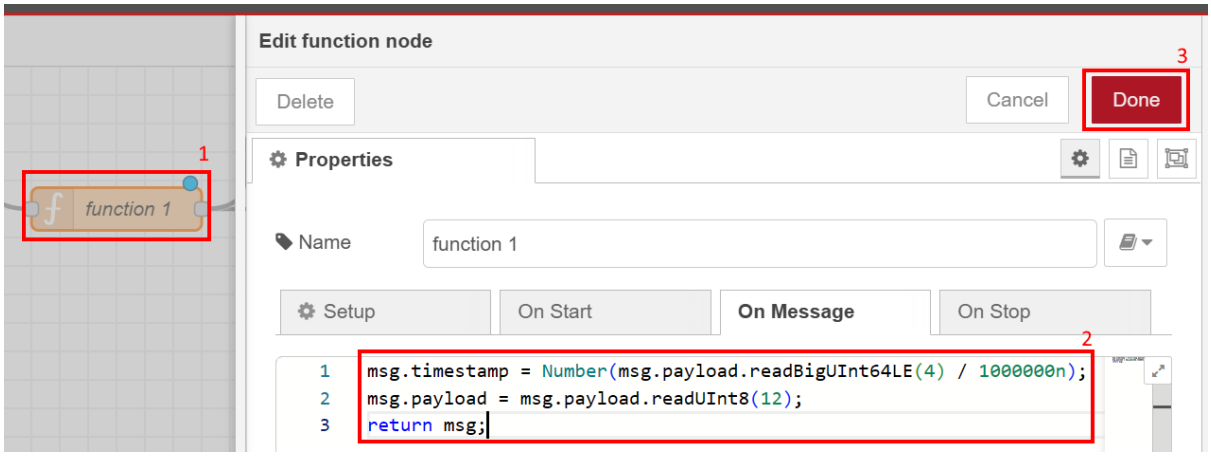


Because the MQTT payload is a raw byte buffer, it must first be processed. The first 4 bytes encode a buffer version. As of writing, the buffer version is 0, followed by an 8 byte timestamp, and finally the variable data.

Double click the function node to open the configuration. The **timestamp** property of the message should be set, and the **payload** property should be replaced with the variable value. The JavaScript buffer offers various functions to read an integer from a buffer at a specified offset. Use a function appropriate for the variable type to turn it into a JavaScript number, see the Node.js documentation (https://nodejs.org/docs/latest/api/buffer.html). Press **Done** to save the changes.

- For 1-byte variables: **readInt8 / readUInt8**
- For 2-byte variables: **readInt16LE / readUInt16LE**
- For 4-byte variables: **readInt32LE / readUInt32LE**
- For 8-byte variables: **readBigInt64LE / readBigUInt64LE**

```
msg.timestamp = Number(msg.payload.readBigUInt64LE(4) / 1000000n);
msg.payload = msg.payload.readUInt8(12);
return msg;
```



`Deploy` the flow and launch the demo. To open the dashboard, use the same base address and port as the Node-RED configurator with the **`/ui`** route. E.g., if the configurator is running on **`http://localhost:1880/#flow/abc`**, open **`http://localhost:1880/ui`**. If the variable changes, the data should be visible in the dashboard near instantly.