Building an OPC UA to LoRaWAN converter

Probably you need to connect industrial controls to LoRaWAN like any OPC UA server from different vendors like Rockwell, Schneider, Siemens, etc. OPC UA is a industry standard used to integrate many devices and software tools like SCADA's etc. You can do this thanks to Node-RED since Node-RED is the universal integration tool. You will use a LoRa radio HAT (Hardware on Top) interface to build your own OPCUA client as a custom-made solution.

In this document we're going to cover the following main topics:

- Creating your own LoRaWAN node with a Raspberry Pi. RFM95 based Lora Pi HAT
- Installing Raspi-LMIC library and setting up your new node on TTN
- Storing data on a binary file and sending it
- Using OPC UA nodes to retrieve data from any OPC UA server in Node-RED
- Reading data from a OPC UA server and sending the data thru LoRaWAN

Technical requirements

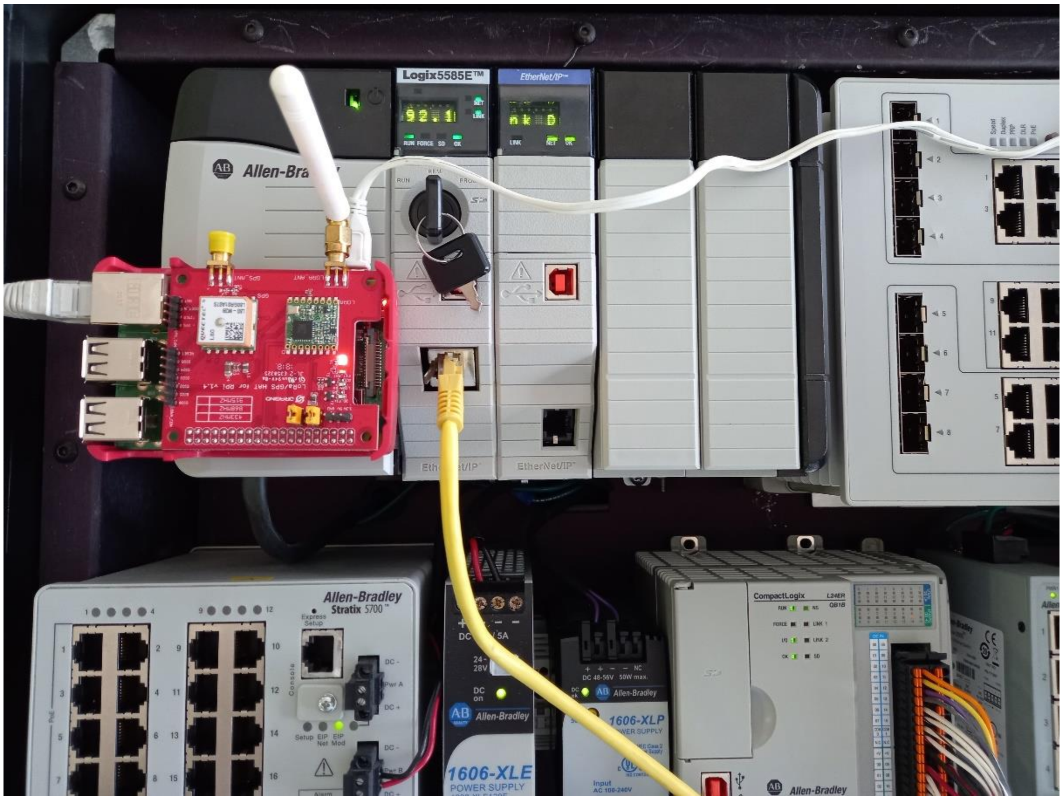
- Basic knowledge on Raspberry Pi
- Basic knowledge of Linux
- Basic knowledge on C or C++ language and how to compile it on a Raspberry Pi
- Studio5000 PLC programming software in case you use a Rockwell PLC (Compactlogix or Controllogix) as EtherNet/IP class 1 data source. (You can use also Micrologix or SLC500 in class 3 EtherNet/IP)

You can find the ttn-otaa.cpp code used here:

https://github.com/xavierflorensa/OPC-UA-to-LoRAWAN

Contents

Here you can see the used testbech or use an emulator like FactoryTalk Logix Echo, or even the data simulator from Kepserver EX.



Creating your own LoRaWAN node with a Raspberry Pi. RFM95 based Lora Pi HAT

In this document you will be able to get EtherNet/IP data from any device (PLC, Drive, Servodrive, etc) and sending thu LoRaWAN.

Or even performing PLC to PLC communications completely in EtherNet/IP thru LoRaWAN as a mesaging service (Considering to have an EtherNet/IP device on each side).

In this document we will use the dragino LoRa Pi HAT attached to a Raspberry Pi, so the Raspberry Pi has an ethernet Port and would be able to send messages to a near gateway. This board is using the RFM95 which is a popular LoRa Chip.

You can find more details on this LoRa Pi HAT here:

https://www.dragino.com/products/lora/item/106-lora-gps-hat.html

Xavier Florensa Automation Specialist Risoul Ibérica

Just plug the LoRa Pi HAT on top of your Raspberry Pi model 3B+ or model 4 and you have already a complete linux based LoRaWAN node, to perform whatever you need, for instance an Ethernet/IP to to LoRaWAN converter.

So now the Raspberry will be the Ethernet to LoRaWAN converter or even better, the EtherNet/IP to LoRaWAN converter.

This is the architecture we will be using.

On the Raspberry Pi will run the OPC UA client and also the OPC UA server in case the PLC does not have it.

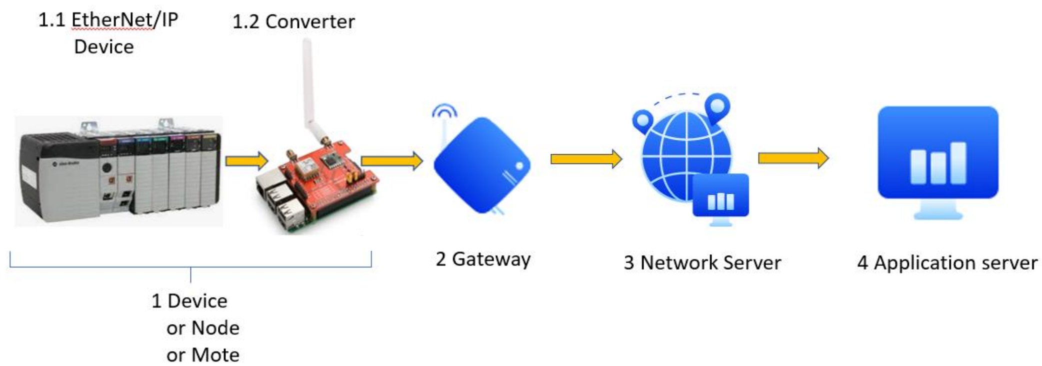


Figure 1 – System architecture used in this document

First of all you need to install the required software as you will see on following section.

Installing Raspi-LMIC library and setting up your new node on TTN

Grab a Raspberry and install Rapberry Pi OS Buster with desktop
Access the desktop and enable ssh Access
Access the Raspberry per SSH
Install node-red from here

https://nodered.org/docs/getting-started/raspberrypi

```
bash <(curl -sL https://raw.githubusercontent.com/node-red/linux-installers/master/deb/update-nodejs-and-nodered)
```

The following versions have been tested

```
pi@raspberrypi:~ $ node -v
v16.16.0
pi@raspberrypi:~ $ npm -v
8.11.0
```

And node-red version 3.0.2

```
v3.0.2
```

Next, follow the explanation:

First you have to Enable SPI interface on your Raspberry Pi, then install wiring Pi with following command:

```
sudo apt-get install wiringpi
```

Then we will follow the steps given on the link below installing the required dependencies.

Be sure to install the command git

```
sudo apt-get install git
```

Then install the software on the Raspberry PI we open the following address on an explorer on our Pi

https://github.com/pmanzoni/raspi-lmic

and clone the repository to your Raspberry Pi

git clone https://github.com/pmanzoni/raspi-lmic.git

```
pi@raspberrypi:~ $ git clone https://github.com/pmanzoni/raspi-lmic.git
Cloning into 'raspi-lmic'...
remote: Enumerating objects: 98, done.
remote: Total 98 (delta 0), reused 0 (delta 0), pack-reused 98
Unpacking objects: 100% (98/98), done.
pi@raspberrypi:~ $
```

Figure 2 – Raspberry Pi terminal with repository clone process

After this the required files are on your Raspberry at this directory /home/pi/raspi-lmic

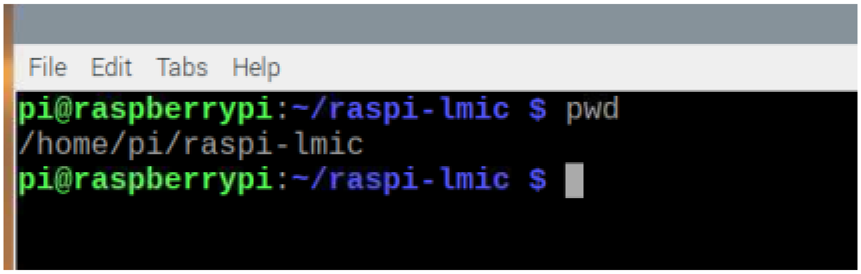


Figure 3 – Raspberry Pi terminal with repository created

Let's take a look at the repository on our Raspberry Pi. There is even a tool to detect the RFM95 module (Or LoRa Pi HAT) called "spi_scan.o". So the RFM95 is detected as you can see on next figure.

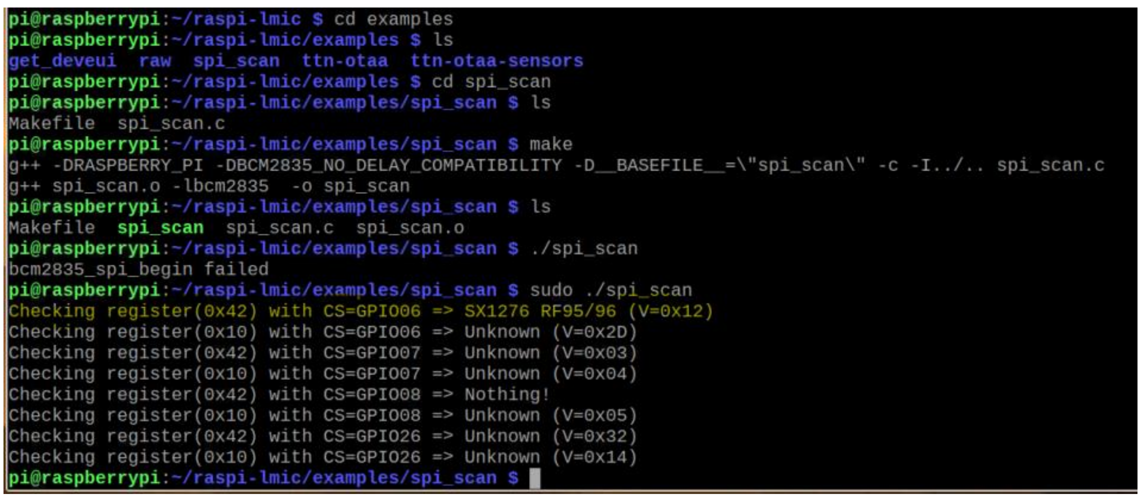


Figure 4 – Raspberry Pi terminal with detected RFM65 module detected

Before compiling the example file ttn-otaa.cpp and executing it let's create a new application on TTN

We will do it manually since we are doing our do it yourself DIY LoRaWAN node so there is not such Pi HAT board on the devices repository.

For AppEUI we can use all zeros.

For the DevEUI let's try with this one { 0x00, 0x04, 0x34, 0x09, 0xf1, 0xeb, 0x27, 0xb8 } obtained from a tool called get_deveui from example repository.

```
pi@raspberrypi:~/raspi-lmic/examples/get_deveui $ sudo ./get_deveui
Use "get_deveui all" to see all interfaces and details
// wlan0 Up Linked TTN Dashboard DEVEUI format B827EBF109340400
static const u1_t PROGMEM DEVEUI[8]={ 0x00, 0x04, 0x34, 0x09, 0xf1, 0xeb, 0x27, 0xb8 }; // wlan0
pi@raspberrypi:~/raspi-lmic/examples/get_deveui $
```

Figure 5 – Raspberry Pi terminal with DevEui key generation

For AppKey, let’s let TTN generate it for us.

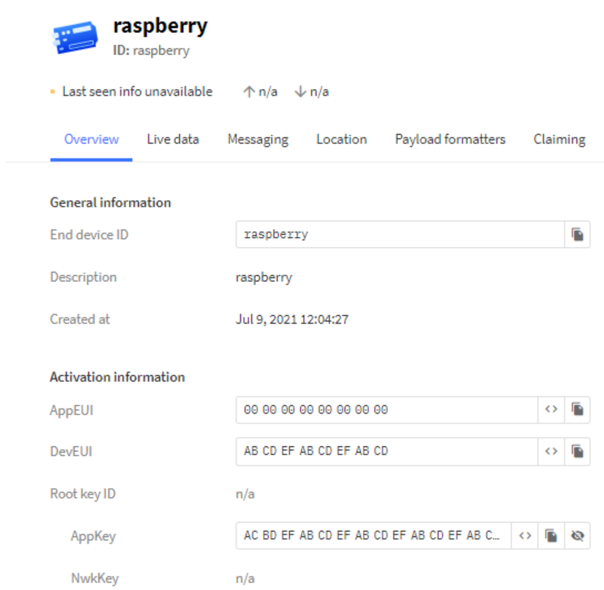


Figure 6 – TTN console with generated AppKey

Let’s modify the credentials AppEui, DevEui and AppKey accordingly on ttn-otaa.cpp file

```
ttn-otaa.cpp

// This EUI must be in little-endian format, so least-significant-byte
// first. When copying an EUI from ttnctl output, this means to reverse
// the bytes. For TTN issued EUIs the last bytes should be 0xD5, 0xB3,0x70.
static const u1_t PROGMEM APPEUI[8]= { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
void os_getArtEui (u1_t* buf) { memcpy_P(buf, APPEUI, 8);}

// This should also be in little endian format, see above.
static const u1_t PROGMEM DEVEUI[8]= { 0xB8, 0x27, 0xEB, 0xF1, 0x09, 0x34, 0x04, 0x00 };
// Here on Raspi we use part of MAC Address do define devEUI so
// This one above is not used, but you can still old method
// reverting the comments on the 2 following line
void os_getDevEui (u1_t* buf) { memcpy_P(buf, DEVEUI, 8);}
//void os_getDevEui (u1_t* buf) { getDevEuiFromMac(buf); }

// This key should be in big endian format (or, since it is not really a
// number but a block of memory, endianness does not really apply). In
// practice, a key taken from ttnctl can be copied as-is.
// The key shown here is the semtech default key.
static const u1_t PROGMEM APPKEY[16] = { 0x3C, 0x42, 0x0D, 0x18, 0xD1, 0x6F, 0x8C, 0xBA, 0x2E, 0x5A, 0x90, 0x19, 0xDE, 0xD0, 0x0A, 0xF1};
void os_getDevKey (u1_t* buf) {  memcpy_P(buf, APPKEY, 16);}

static uint8_t mydata[] = "Raspi TESTING!";
```

Figure 7 – Example code on Raspberry Pi with the TTN Keys used in this chapter

Let’s compile now the ttn-otaa.cpp file (just a test text string “Raspi TESTING!”)

```
pi@raspberrypi:~/raspi-lmic/examples/ttn-otaa $ make
g++ -std=c++11 -DRASPBERRY_PI -DBCM2835_NO_DELAY_COMPATIBILITY -D__BASEFILE__=\"ttn-otaa\" -c -I../../src  ttn-otaa.cpp
g++ ttn-otaa.o raspi.o radio.o oslmic.o lmic.o hal.o aes.o -lbcm2835 -o ttn-otaa
```

Figure 8 – Example code on Raspberry Pi with the TTN Keys used in this chapter

Now we can run the executable file “ttn-otaa.o” using sudo.

```
pi@raspberrypi:~/raspi-lmic/examples/ttn-otaa $ sudo ./ttn-otaa
ttn-otaa Starting
RFM95 device configuration
CS=GPIO25 RST=GPIO17 LED=Unused DIO0=Unused DIO1=Unused DIO2=Unused
DevEUI : 00043409F1EB27B8
AppEUI : 0000000000000000
AppKey : 3C420D18D16F8CBA2E5A9019DED00AF1
18:35:26: Packet queued
18:35:26: EV_JOINING
18:35:36: EV_JOINED
18:35:41: EV_TXCOMPLETE (includes waiting for RX windows)
18:36:41: Packet queued
18:36:52: EV_TXCOMPLETE (includes waiting for RX windows)
18:37:52: Packet queued
18:38:02: EV_TXCOMPLETE (includes waiting for RX windows)
18:39:02: Packet queued
18:39:11: EV_TXCOMPLETE (includes waiting for RX windows)
18:40:11: Packet queued
```

Figure 9 – Raspberry Pi terminal with execution of example

On TTN console we are happy to see our just created LoRaWAN node (Raspberry Pi) sending uplink test messages.

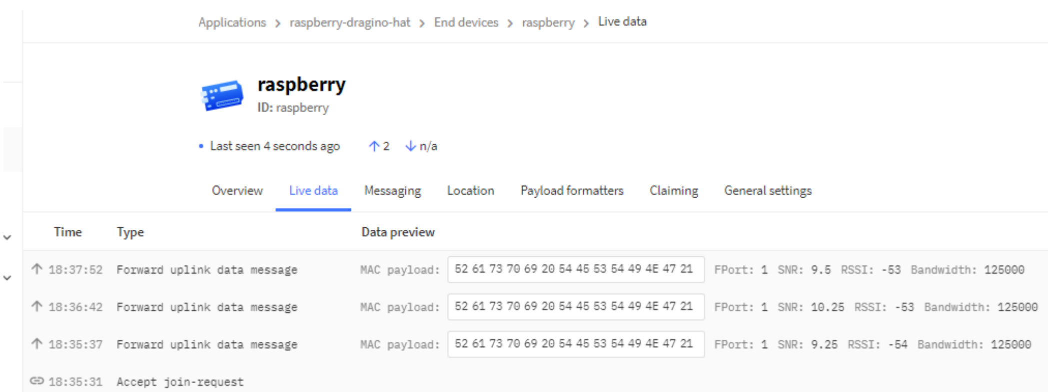


Figure 10 – TTN console with Raspberry Pi test transmitted uplink messages

It Works!!

Now we try to decode the payload with payload formatter given on the repository

```
function Decoder(bytes, port) {
  // Decode plain text; for testing only
  return {
      myTestValue: String.fromCharCode.apply(null, bytes)
  };
}
```

And this is the decoded payload on TTN console

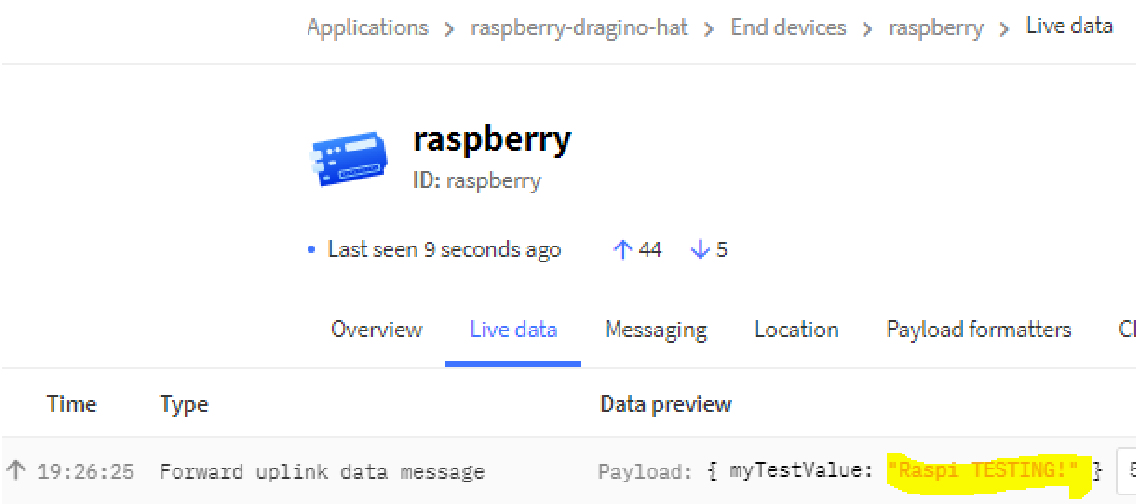


Figure 11 – TTN console with Raspberry Pi test decoded uplink messages

You can find the ttn-otaa.cpp code used here:

https://github.com/xavierflorensa/OPC-UA-to-LoRAWAN

Next we will create a text file to store the data we want to send thru LoRaWAN. These data on the file will be later on written by the OPC server thru Node-RED.

Storing data on a binary file and sending it

Now let's try to modify the program to get the data with a file from OPC UA. We will try to use the code as a on the example

Read Write Binary File

```
  //OPEN CONFIG FILE IN OUR APPLICAITONS DIRECTORY OR CREATE IT IF IT DOESN'T EXIST
  FILE *file1;
  unsigned char file_data[100];
  const char *filename1 = "config.conf";

  file1 = fopen(filename1, "rb");
  if (file1)
  {
    //----- FILE EXISTS -----
    fread(&file_data[0], sizeof(unsigned char), 100, file1);

    printf("File opened, some byte values: %i %i %i %i\n", file_data[0], file_data[1],
file_data[2], file_data[3]);

    fclose(file1);
    file1 = NULL;
  }
```

Figure 12 – Example on how to read write a binary file

So we modify again the ttn-otaa.cpp from last section file like this way

Except the last two lines (we do not erase the file, if you erase the file the program execution breaks)

First of all let's create a file called missatge.txt on the same directory where we have the executable like: /home/pi/raspi-lmic/examples/ttn-otaa

So instead we change the ttn-otaa.cpp like this. The code we add to the last section tth-otaa.cpp is the one between the commented lines //*****

```
void do_send(osjob_t* j) {
    char strTime[16];
    getSystemTime(strTime , sizeof(strTime));
    printf("%s: ", strTime);

    // Check if there is not a current TX/RX job running
    if (LMIC.opmode & OP_TXRXPEND) {
        printf("OP_TXRXPEND, not sending\n");
    } else {
        digitalWrite(RF_LED_PIN, HIGH);
        // Prepare upstream data transmission at the next possible time.
        //*************************************************************************************

        int c; char* cstr; FILE * missatge;
        missatge = fopen("missatge.txt","r");
        char mystring [100];
        if (missatge == NULL) perror ("Error opening file");
        else
            { if ( fgets (mystring , 100 , missatge) != NULL ){ puts (mystring);}
            }
        fclose(missatge);
        char buf[100];
        int i=0;
        sprintf(buf,mystring, cstr++);
        while(buf[i])
            {
                mydata[i]=buf[i];
                i++;
            }
        mydata[i]='\0';
        LMIC_setTxData2(1, mydata, strlen(buf), 0);
        //remove("missatge.txt");
        //*************************************************************************************
        //LMIC_setTxData2(1, mydata, sizeof(mydata)-1, 0);
        printf("Packet queued\n");
    }
    // Next TX is scheduled after TX_COMPLETE event.
}
```

Figure 13 – Modified ttn_otaa.cpp code with file read

You can find the ttn-otaa.cpp code used here:

https://github.com/xavierflorensa/OPC-UA-to-LoRAWAN

Then we can compile it again.

Then we fill the file with a new message like “hello world”

Let’s inject a new data on the file with the help of Node-RED, as this is the same environment we will use later on when we will extract the data from an OPC UA server.

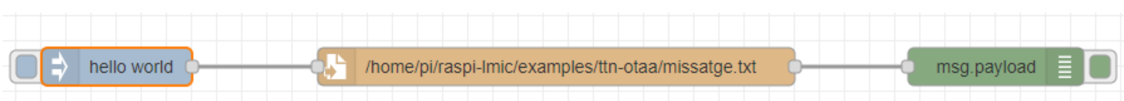


Figure 14 – Node-RED flow to inject data on file missatge.txt

Configure the inject node this way

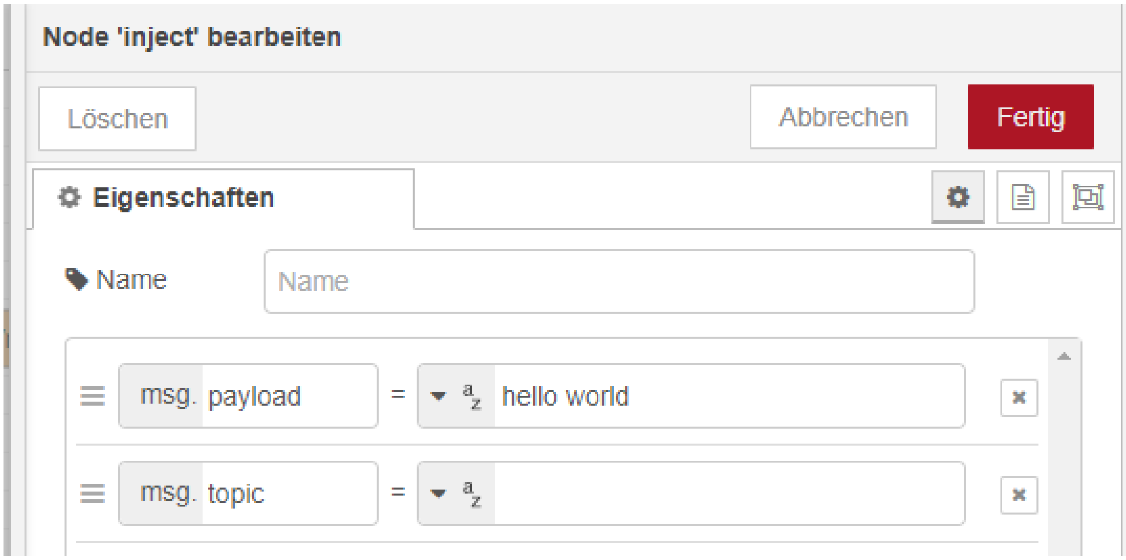


Figure 15 – Node-RED inject node configuration

Then configure the write to file node on this way

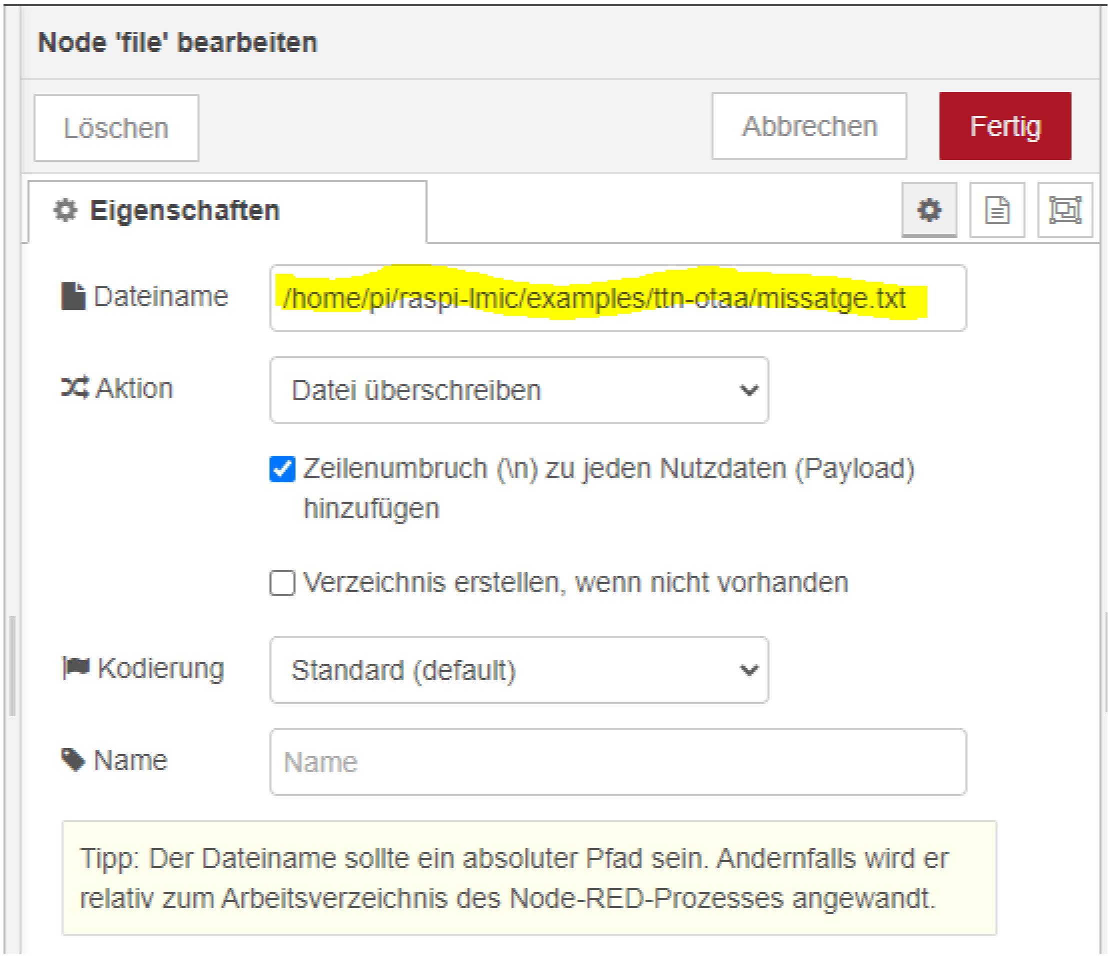


Figure 16 – Node-RED write file node configuration

And test it.

It Works !! As soon as you inject any content on the file, it will be displayed on TTN console, provided that application ttn-otaa is running simultaneously on your Raspberry PI.

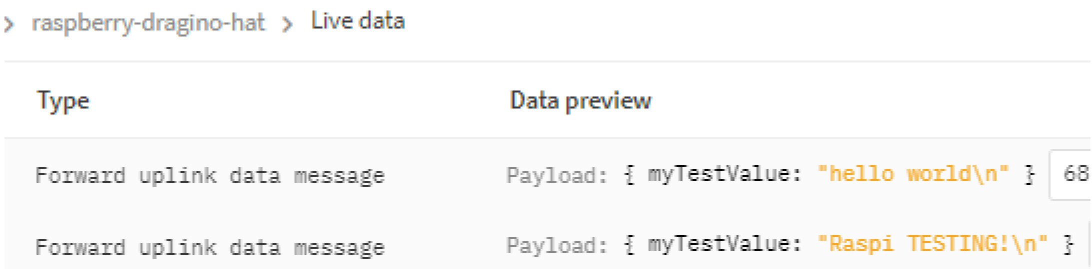


Figure 17 – Node-RED write file node configuration

You can also see the results on Raspberry Pi terminal

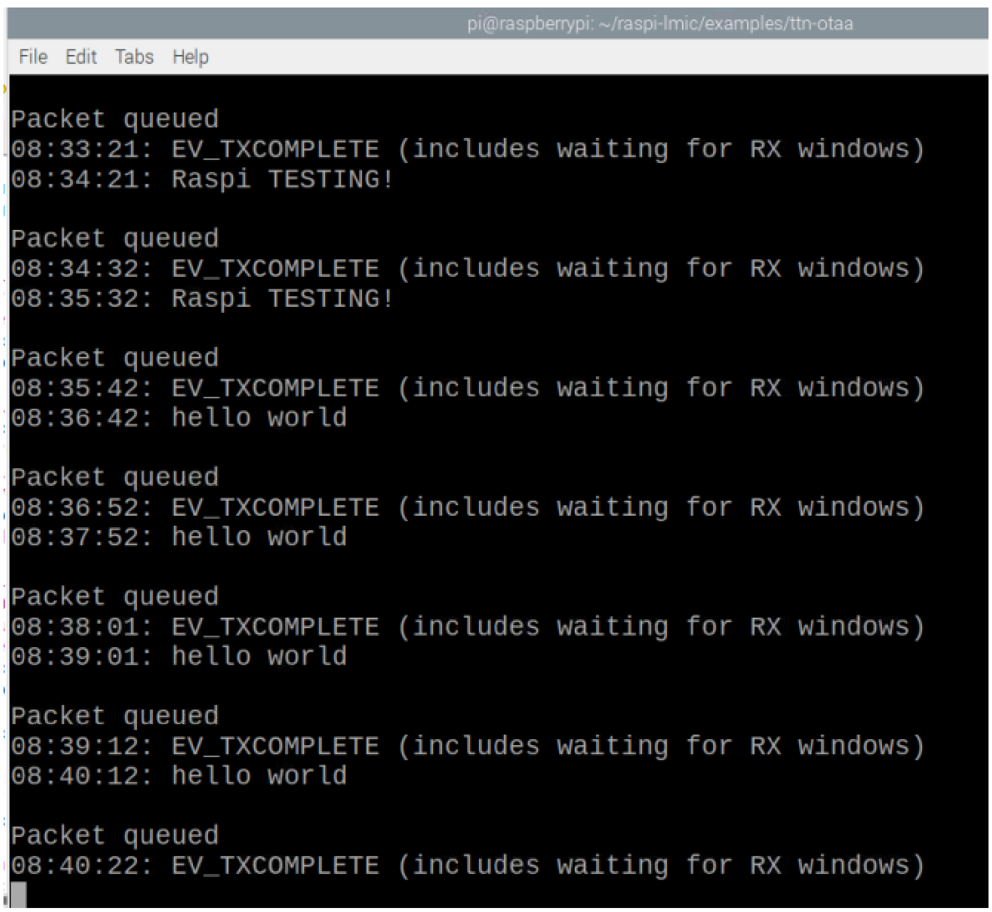
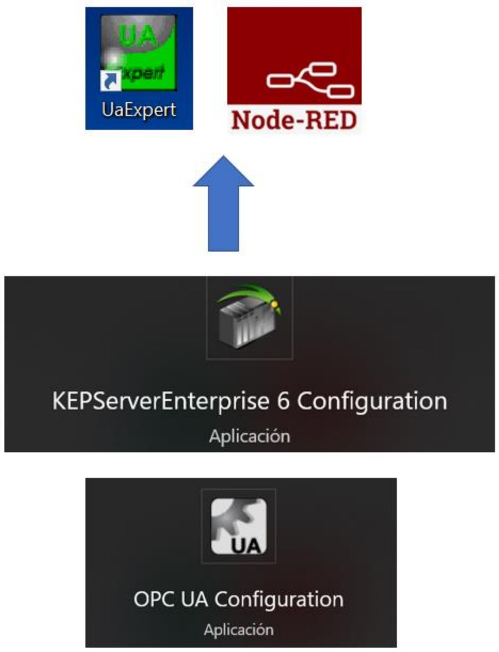


Figure 18 – Raspberry Pi terminal with transmitted messages

Now that you have the C++ software ready for it’s job, you can focus on how to get data from an OPC UA server and inject it on the bin file missatge.txt

Using OPC UA client node to retrieve data from your OPC server in Node-RED

Part one: Kepserver Enterprise as OPC UA server



Third-party OPC UA client software

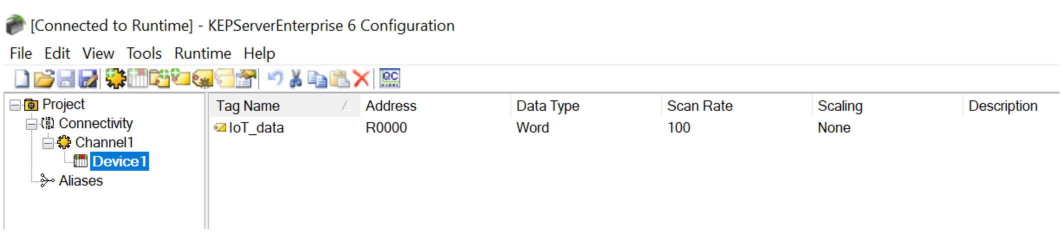
Tags
Data simulator

Server Port
Trusted clients

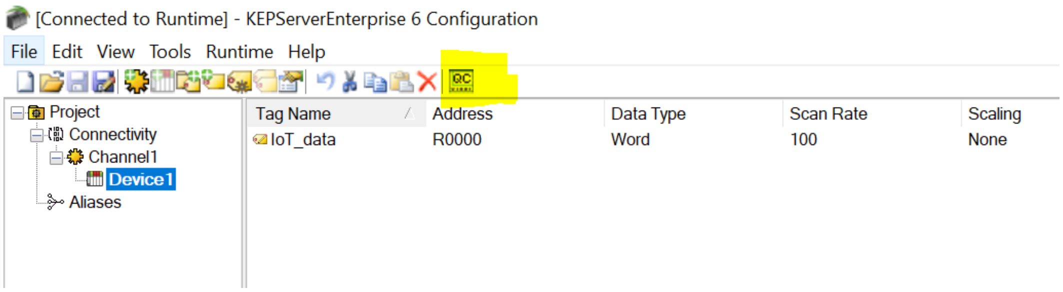
First of all we have to test if we are able to connect to this server

We will test the simulator included with Kepserver EX

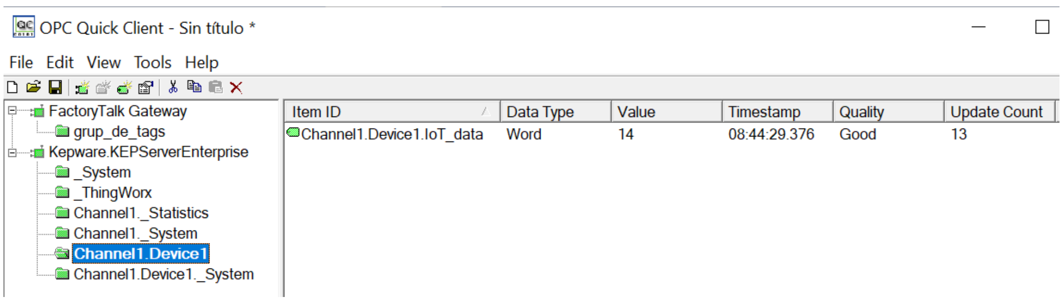
Let’s create a variable



Then test with integrated OPC Quick Client



Yes the variabe is changing



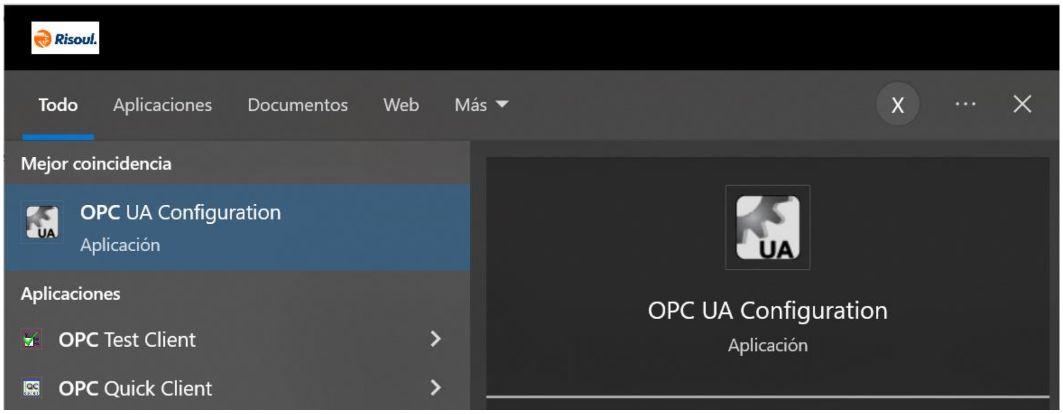
But we want to access that server from an OPC client.

We do not know the port

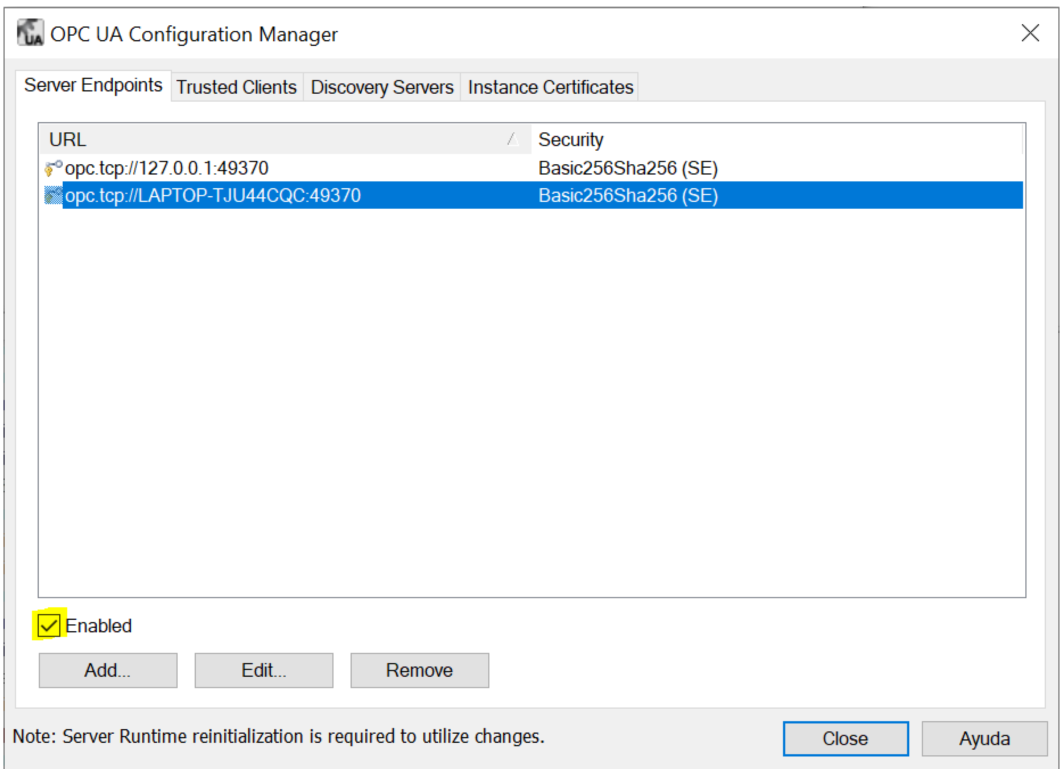
This is explained on this video

https://www.youtube.com/watch?v=pumlhz_h0Qs

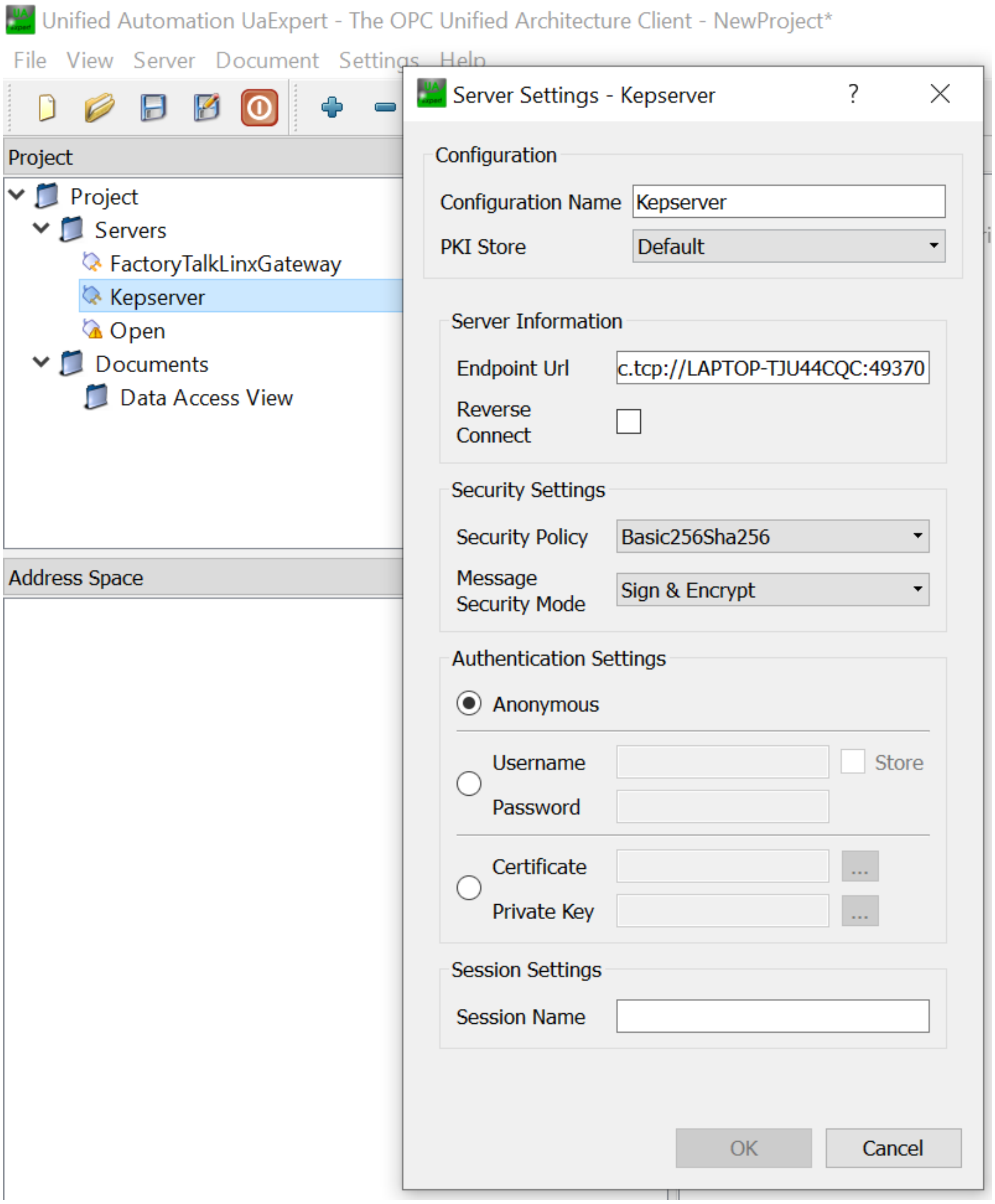
Open OPC configurator



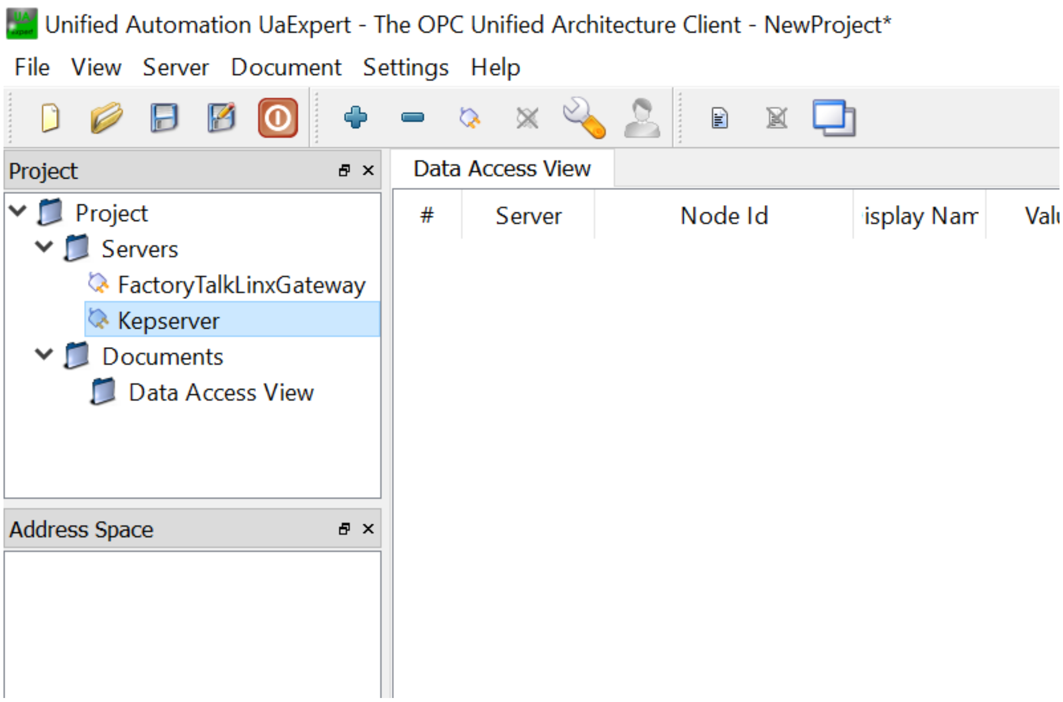
Here you can see the port: 49370, let's enable the second connection



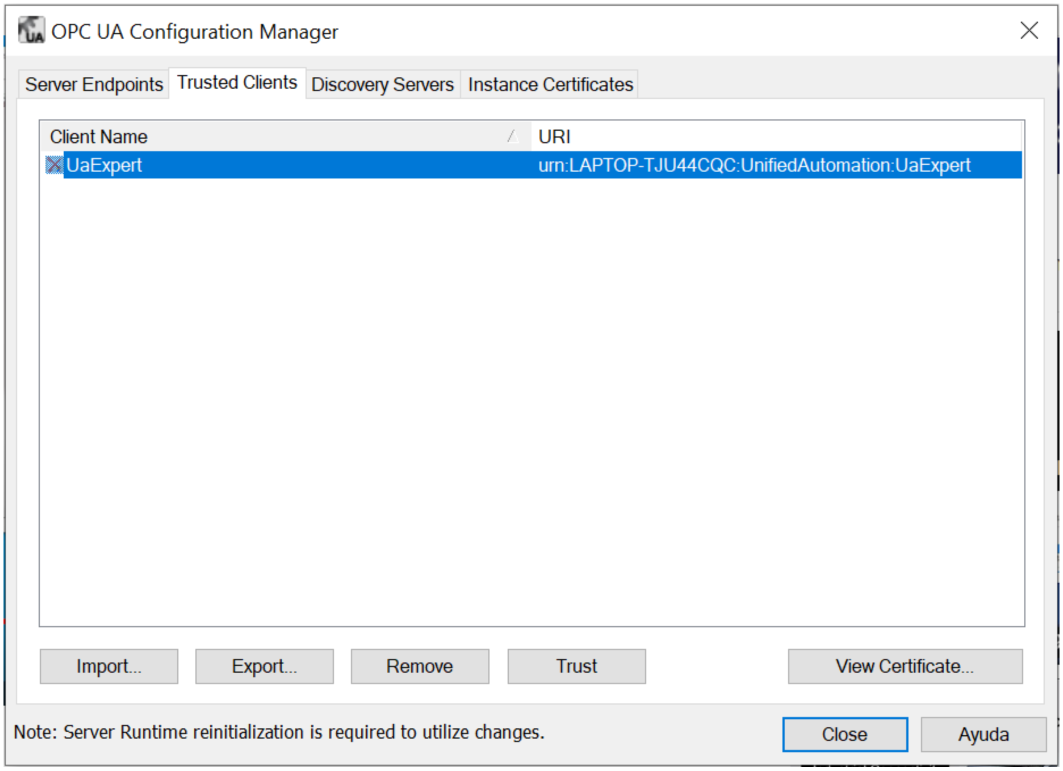
Now let's go to UA Expert OPC client and add a new server with the localhost address and the port


Unified Automation UaExpert - The OPC Unified Architecture Client - NewProject*
File View Server Document Settings Help
Project
Project
Servers
FactoryTalkLinxGateway
Kepserver
Open
Documents
Data Access View
Address Space
Server Settings - Kepserver
Configuration
Configuration Name
Kepserver
PKI Store
Default
Server Information
Endpoint Url
c.tcp://LAPTOP-TJU44CQC:49370
Reverse Connect
Security Settings
Security Policy
Basic256Sha256
Message Security Mode
Sign & Encrypt
Authentication Settings
Anonymous
Username
Store
Password
Certificate
Private Key
Session Settings
Session Name
OK
Cancel

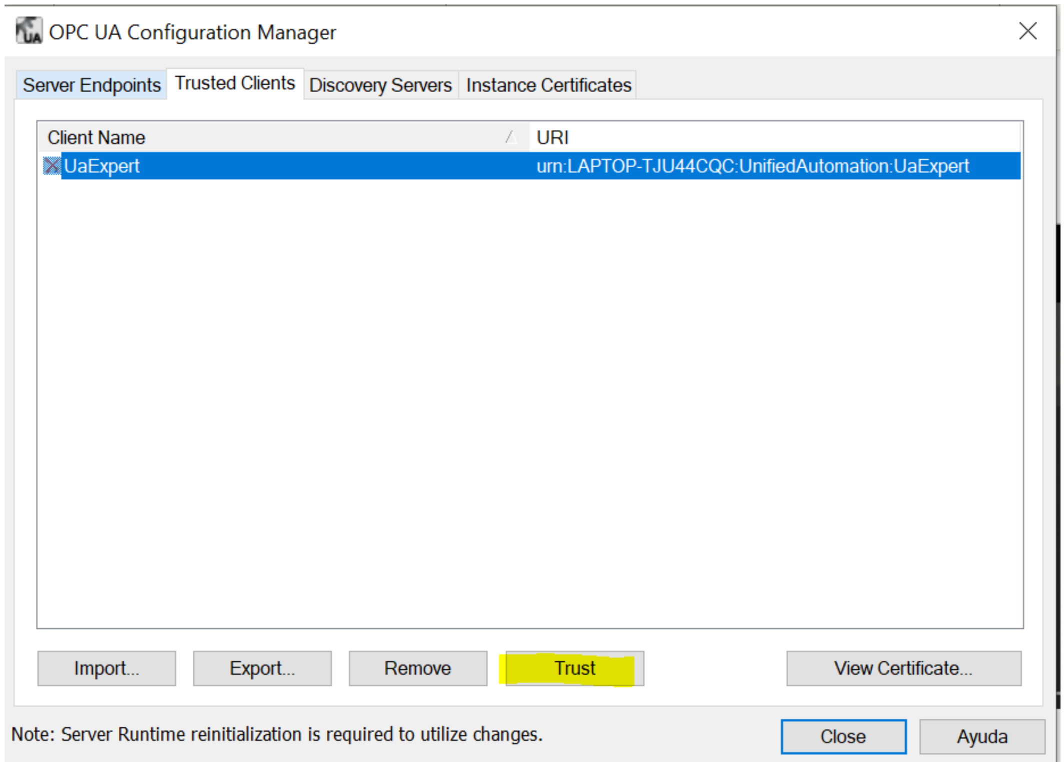
And first time this will not work since the client is not trusted

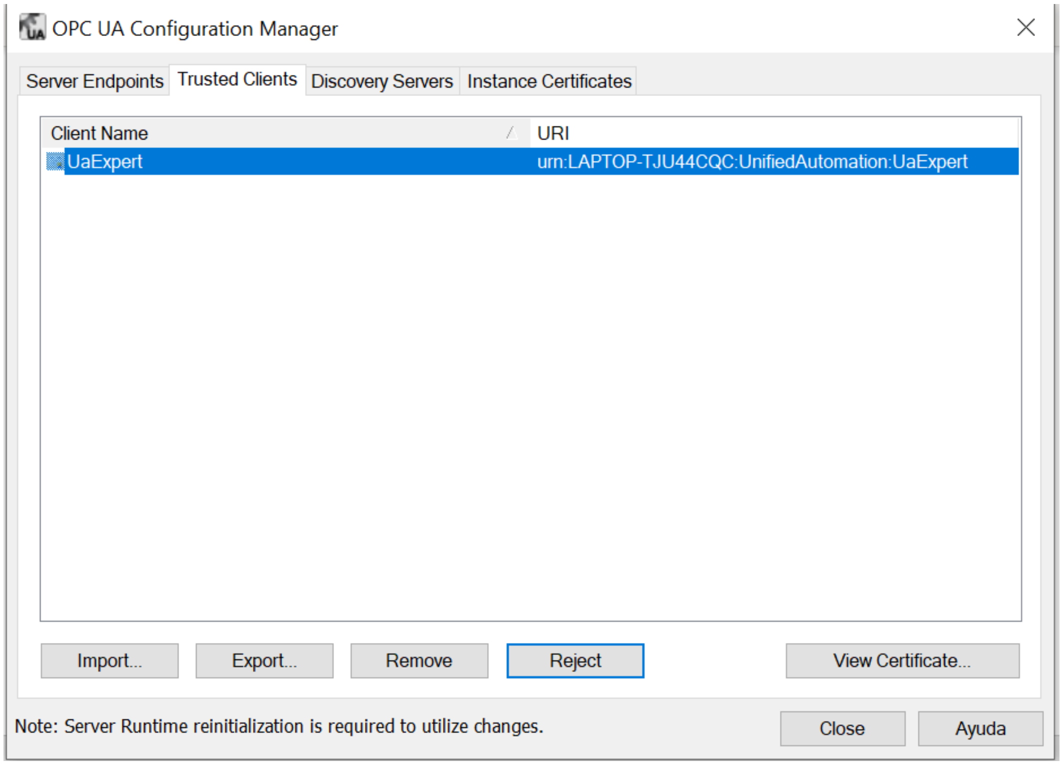


Now let's go again to OPC configuration / Trusted clients

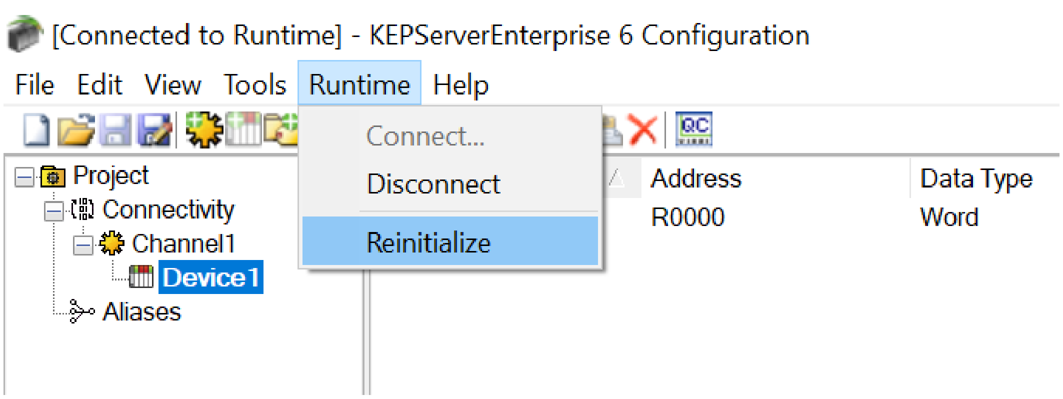


Click on Trust


OPC UA Configuration Manager
Server Endpoints
Trusted Clients
Discovery Servers
Instance Certificates
Client Name
URI
UaExpert
urn:LAPTOP-TJU44CQC:UnifiedAutomation:UaExpert
Import...
Export...
Remove
Trust
View Certificate...
Note: Server Runtime reinitialization is required to utilize changes.
Close
Ayuda

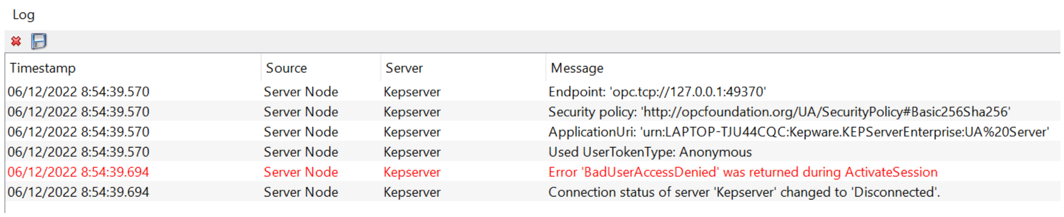


Now we have to reinitialize kepware



Now go again to UA Expert OPC UA client

Still is not connecting,



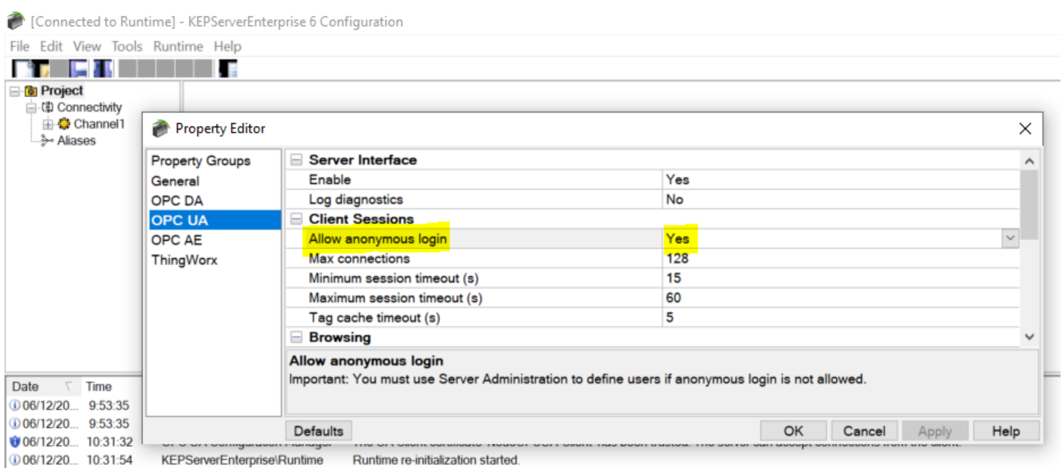
Log

Timestamp	Source	Server	Message
06/12/2022 8:54:39.570	Server Node	Kepserver	Endpoint: 'opc.tcp://127.0.0.1:49370'
06/12/2022 8:54:39.570	Server Node	Kepserver	Security policy: 'http://opcfoundation.org/UA/SecurityPolicy#Basic256Sha256'
06/12/2022 8:54:39.570	Server Node	Kepserver	ApplicationUri: 'urn:LAPTOP-TJU44CQC:Kepware.KEPServerEnterprise:UA%20Server'
06/12/2022 8:54:39.570	Server Node	Kepserver	Used UserTokenType: Anonymous
06/12/2022 8:54:39.694	Server Node	Kepserver	Error 'BadUserAccessDenied' was returned during ActivateSession
06/12/2022 8:54:39.694	Server Node	Kepserver	Connection status of server 'Kepserver' changed to 'Disconnected'.

We have to check following items:

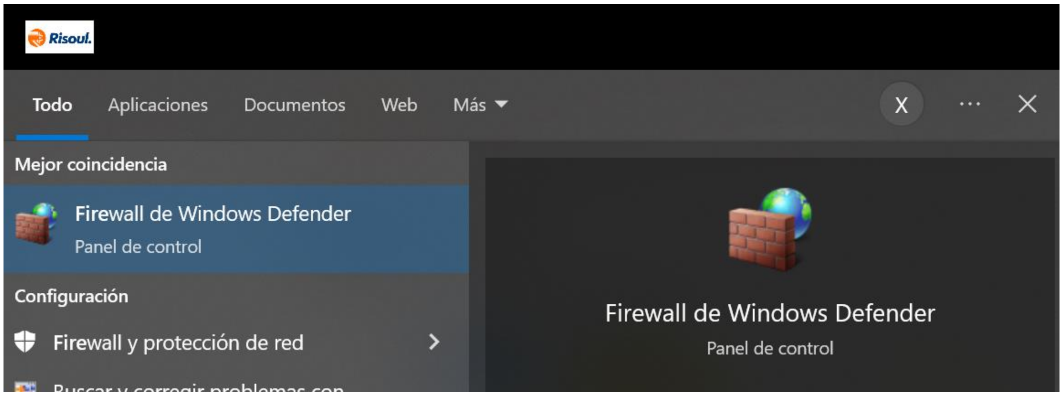
- Accept Anonymous connections (Since Kepserver EX does not accept Anonymous connections by default)
- Setup Windows Firewall

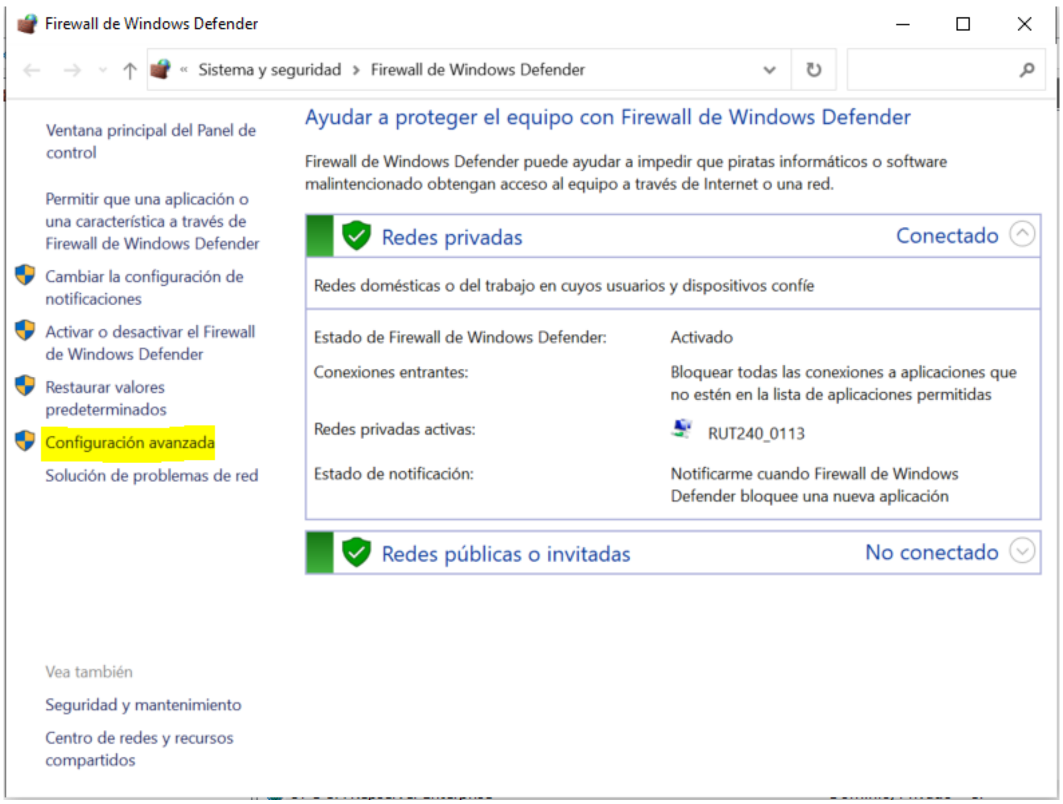
To accept Anonymous connections let’s go to Kepserver Enterprise



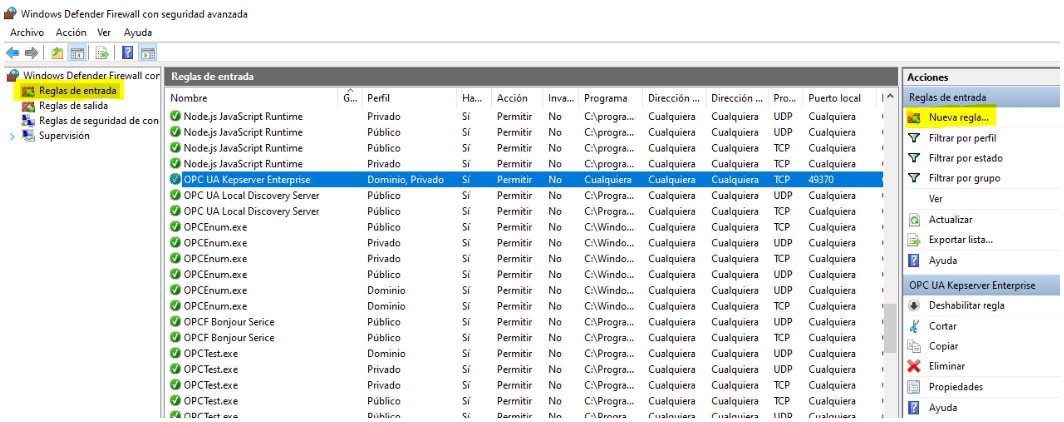
To settle Windows Firewall

Let’s create a new entry





New rule with a new name, for instance: OPC UA Kepserver Enterprise



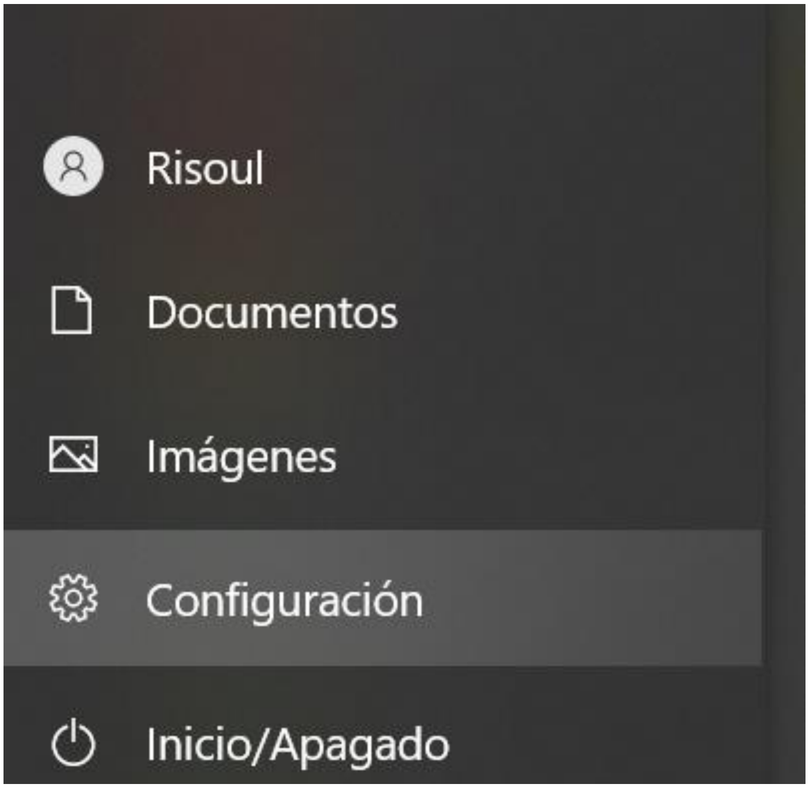
Take a look at our rule, no public (no internet)

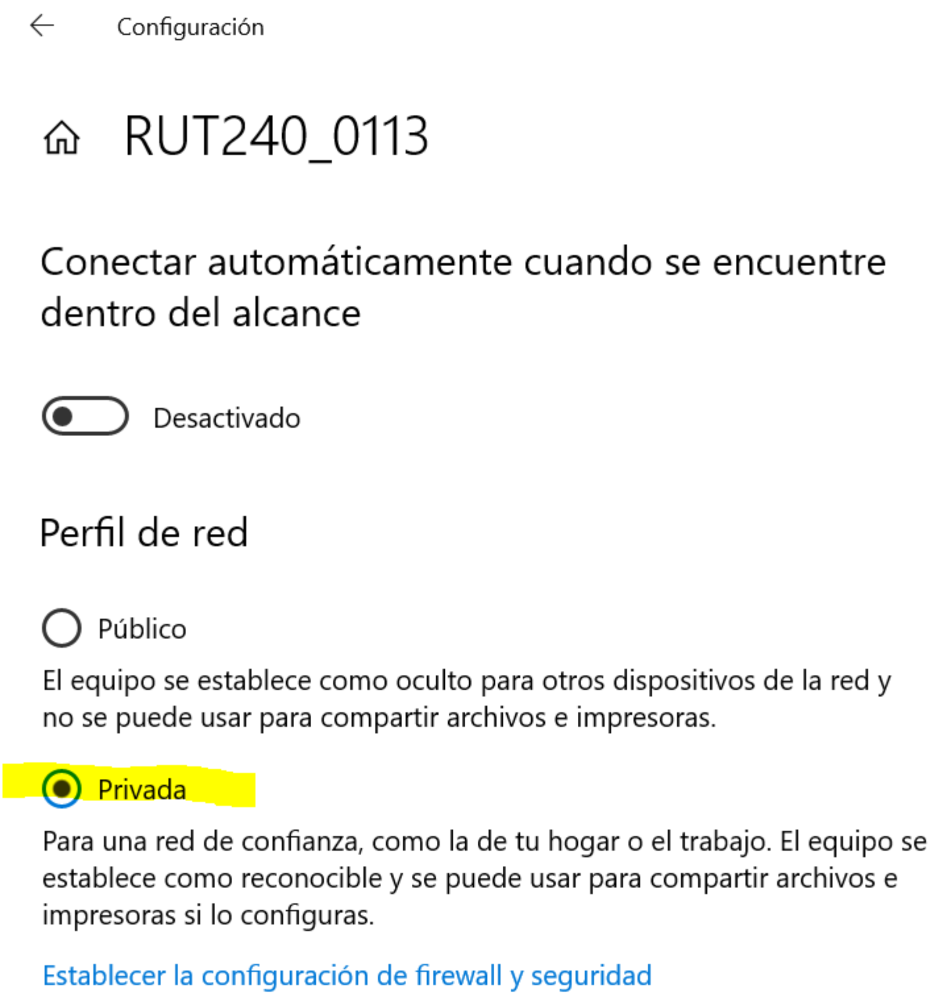
Still the client was not connecting.

This is since I was using a private network, a router to give the Raspberry access to my computer, with my computer connected thru wifi to the router, and the Raspberry thru cable to the router.

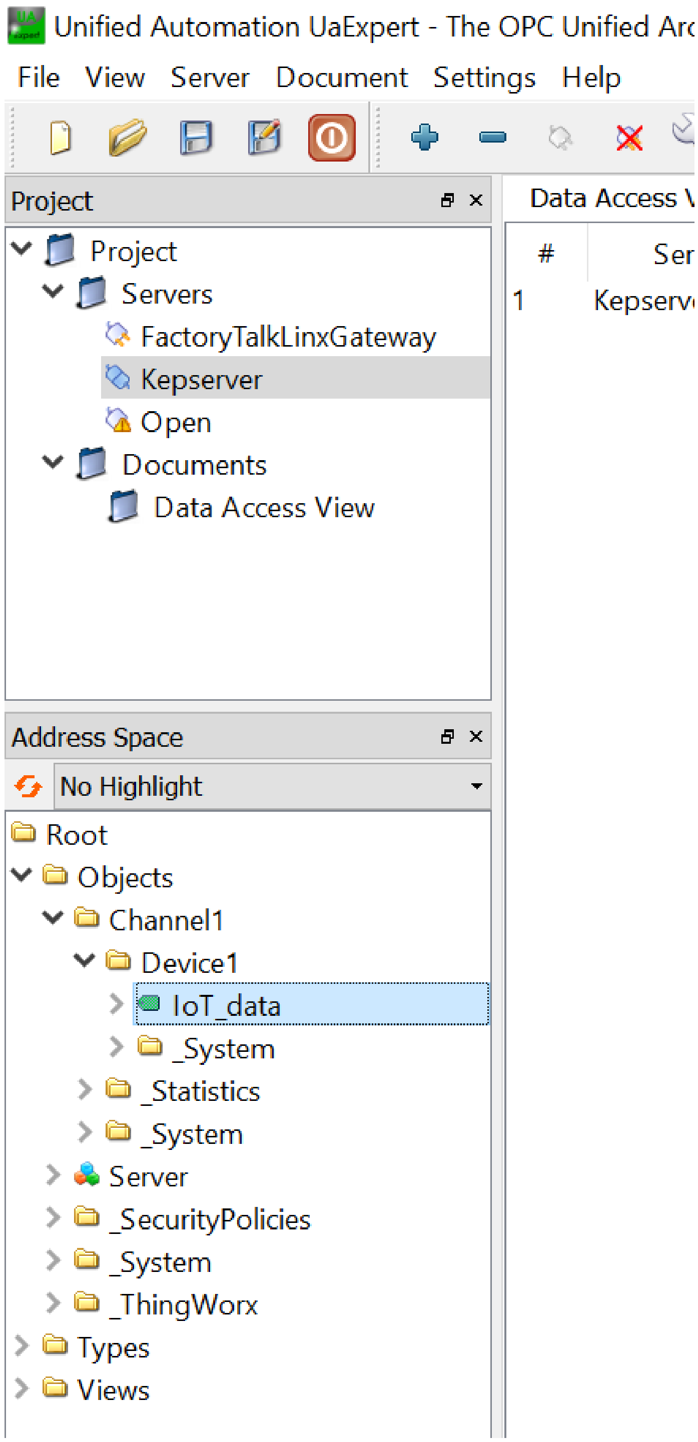
I have to change the network from public to private

Configuración


Risoul
Documentos
Imágenes
Configuración
Inicio/Apagado

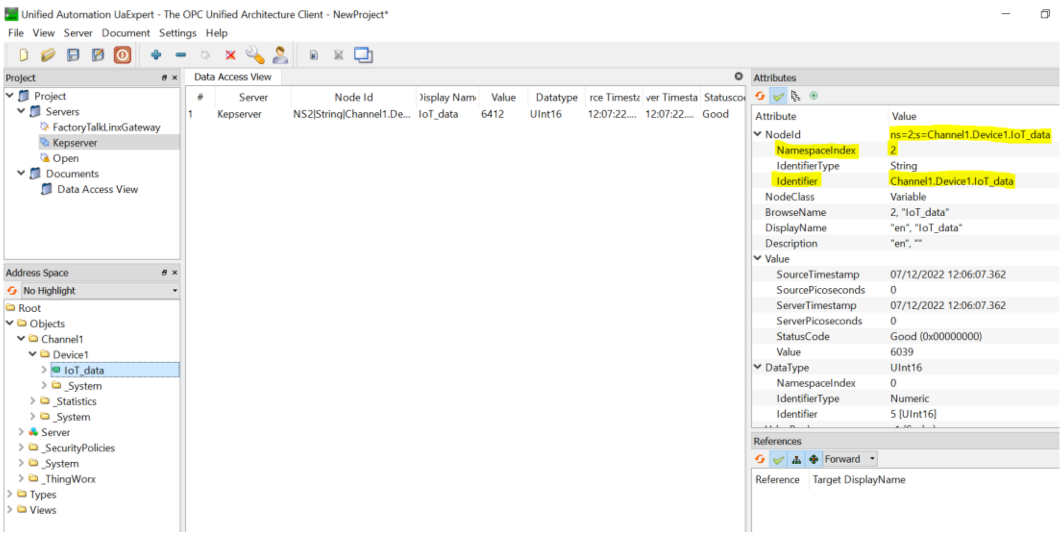


Now it is working

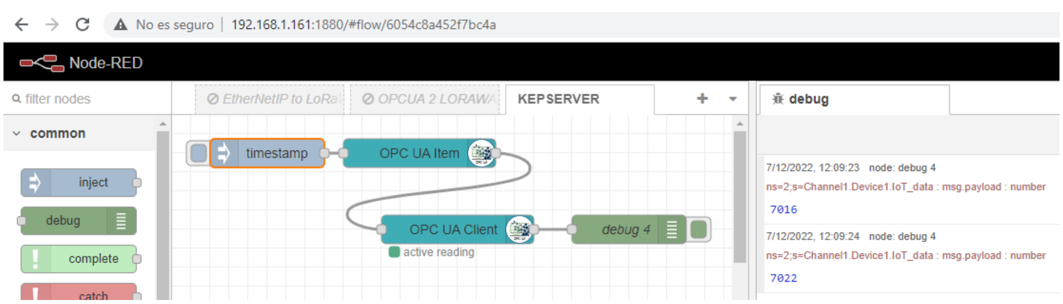

Unified Automation UaExpert - The OPC Unified Ar
File View Server Document Settings Help
Project
Project
Servers
FactoryTalkLinxGateway
Kepserver
Open
Documents
Data Access View
Data Access \
#
Ser
1
Kepserv
Address Space
No Highlight
Root
Objects
Channel1
Device1
IoT_data
_System
_Statistics
_System
Server
_SecurityPolicies
_System
_ThingWorx
Types
Views

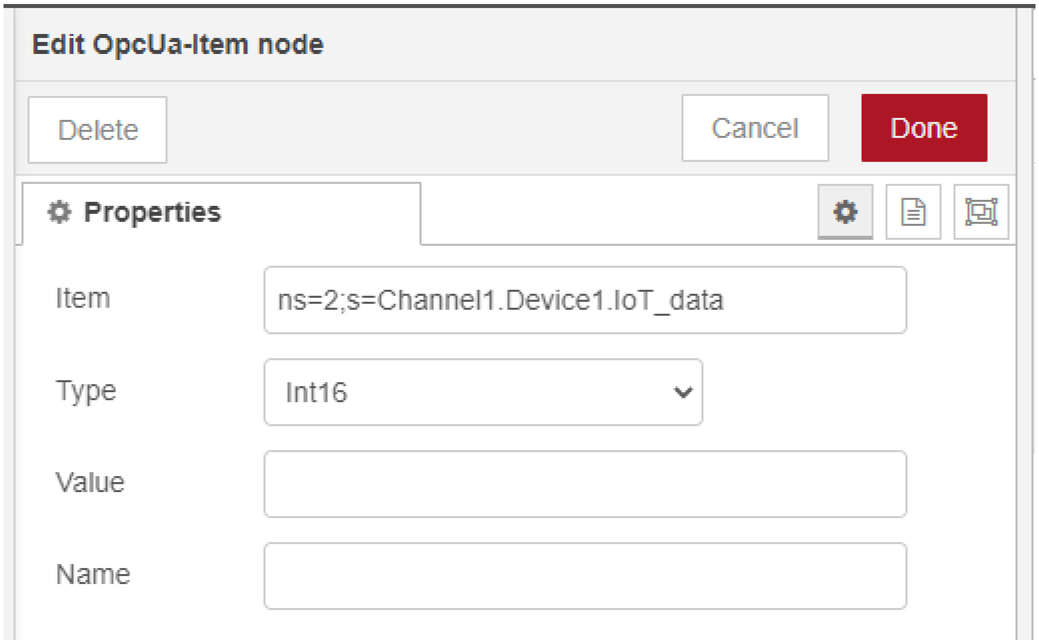
Drag and drop the variable to the workspace, and copy the identifier since you will use later on Node-RED

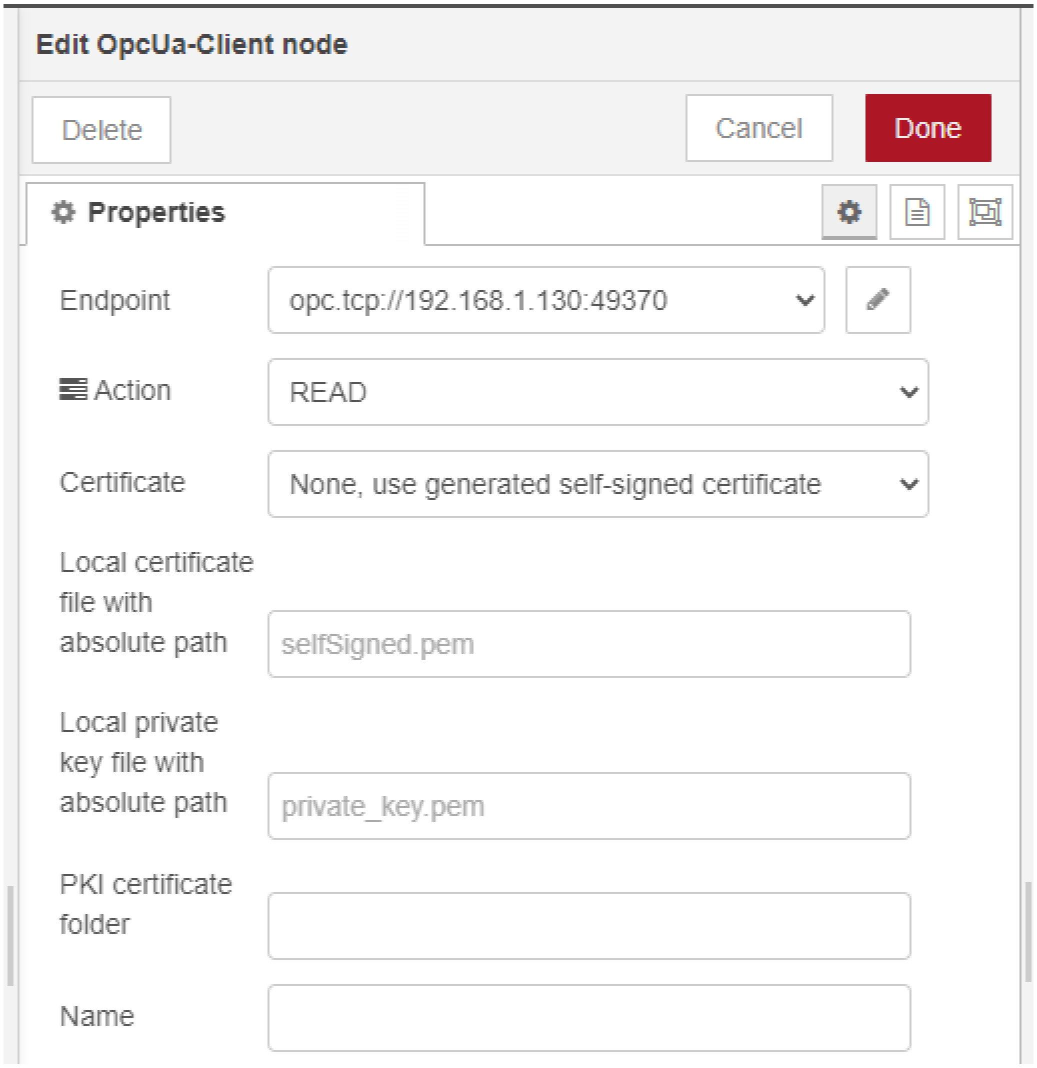
ns=2;s=Channel1.Device1.IoT_data

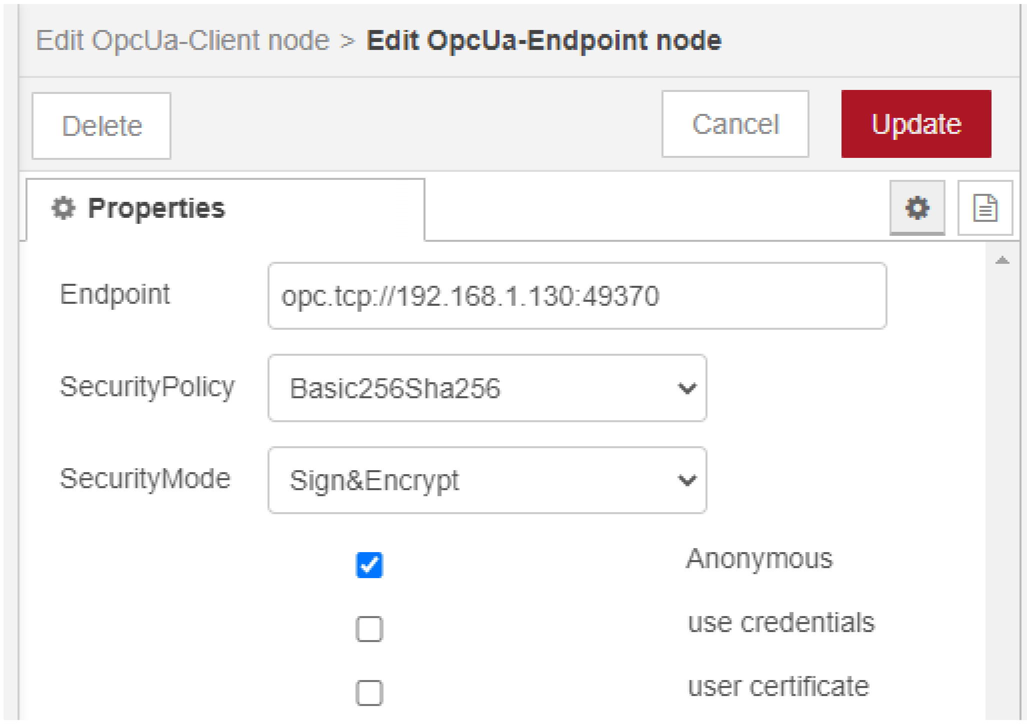


Now let's go to Node-RED on our Raspberry with Lora Pi-HAT

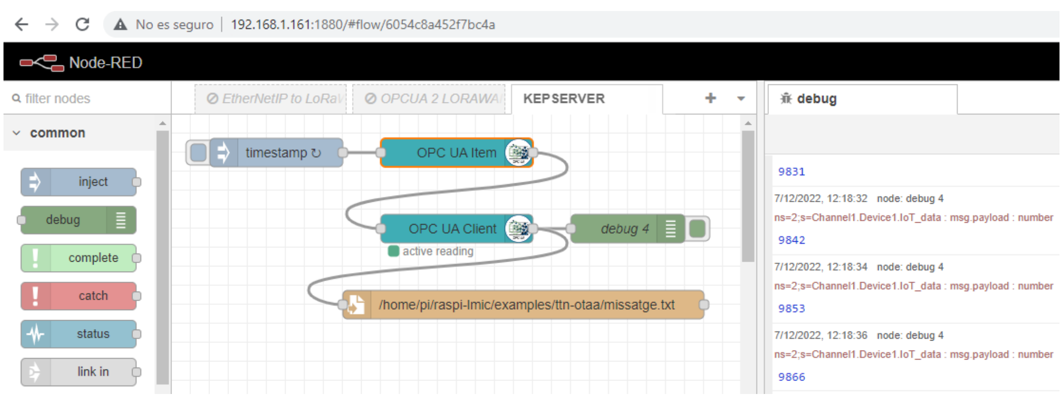



Edit OpcUa-Item node
Delete
Cancel
Done
Properties
Item
ns=2;s=Channel1.Device1.IoT_data
Type
Int16
Value
Name


Edit OpcUa-Client node
Delete
Cancel
Done
Properties
Endpoint
opc.tcp://192.168.1.130:49370
Action
READ
Certificate
None, use generated self-signed certificate
Local certificate file with absolute path
selfSigned.pem
Local private key file with absolute path
private_key.pem
PKI certificate folder
Name



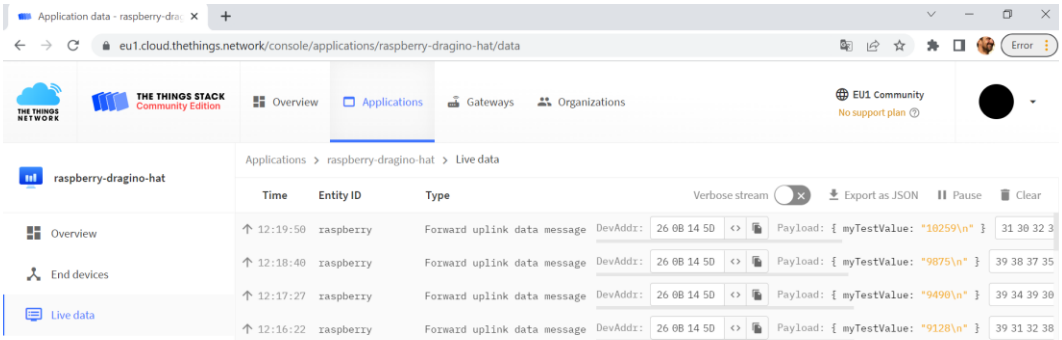
Now let's write on the file each 2 seconds



```
pi@raspberrypi: ~/raspi-lmic/examples/ttn-otaa
12228
^C
pi@raspberrypi:~/raspi-lmic/examples/ttn-otaa $ tail -n 1 -s 2 -F missatge.txt
12338
tail: missatge.txt: file truncated
12349
tail: missatge.txt: file truncated
12360
```

So we are sending the Kepserver simulator data to LoRaWAN

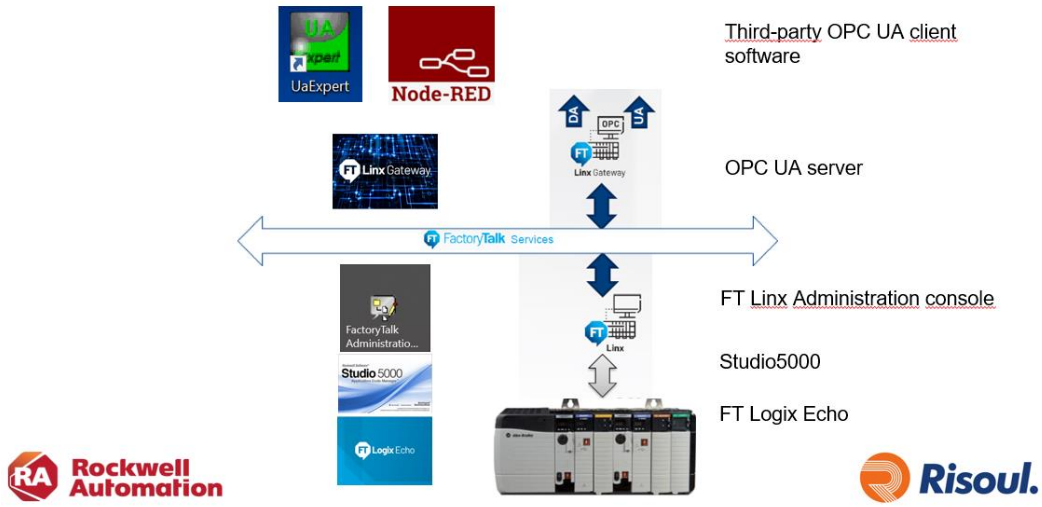
```
pi@raspberrypi: ~/raspi-lmic/examples/ttn-otaa
Packet queued
12:15:22: EV_TXCOMPLETE (includes waiting for RX windows)
12:16:22: 9128

Packet queued
12:16:27: EV_TXCOMPLETE (includes waiting for RX windows)
12:17:27: 9490

Packet queued
12:17:38: EV_TXCOMPLETE (includes waiting for RX windows)
12:18:38: 9875

Packet queued
12:18:47: EV_TXCOMPLETE (includes waiting for RX windows)
12:19:47: 10259

Packet queued
12:19:58: EV_TXCOMPLETE (includes waiting for RX windows)
```



You can find the node-red flow here

https://github.com/xavierflorensa/OPC-UA-to-LoRAWAN/blob/main/OPCUA%20to%20LoRaWAN%20node-red%20flow.json

Part two: Rockwell Automation environment



We need to prepare the OPC UA client node to extract the data from a device thru EtherNet TCP/IP and send it thru LoRaWAN

But before this we have to setup the OPC UA server. In this document we will use a Rockwell Automation environment. A PLC with Ethernet Port like the Controllogix L85E. A OPC UA server like the Factory Talk Linx Gateway. Let's prepare the PLC.

You can find the PLC program used here

https://github.com/xavierflorensa/EtherNet_IP-to-LoRaWAN-converter/blob/main/IoT_sinewave_TTN.ACD

Open Studio5000 and use this settings for Ethernet PLC IP address: 192.168.1.2

Since we need to know the fixed PLC IP address in order to point to it from Node-RED.

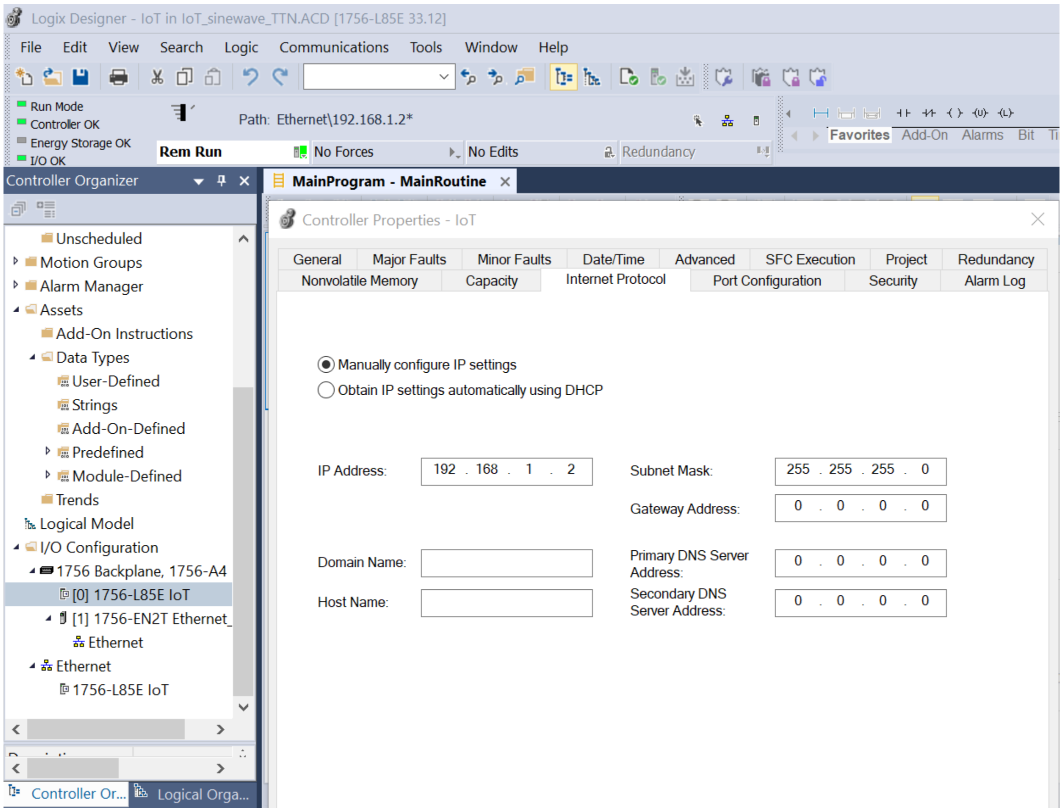


Figure 19 – Studio5000 screenshot with Ethernet configuration

We create a Tag type Double INT to store our data, any value, even digital inputs status or whatever data you want. For instance IoT_data that will store a value changing between 0 and 200.

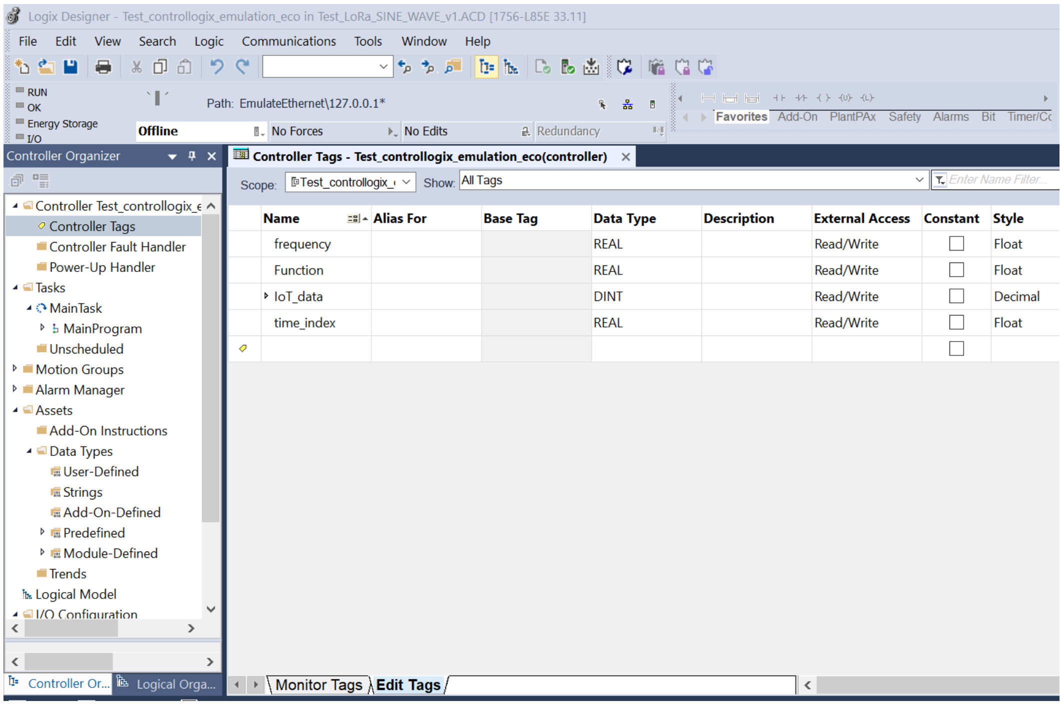


Figure 20 – Studio5000 screenshot with Tag creation

We prepare a program just to IoT_data that will store a value changing between 0 and 200 following a sine wave law.

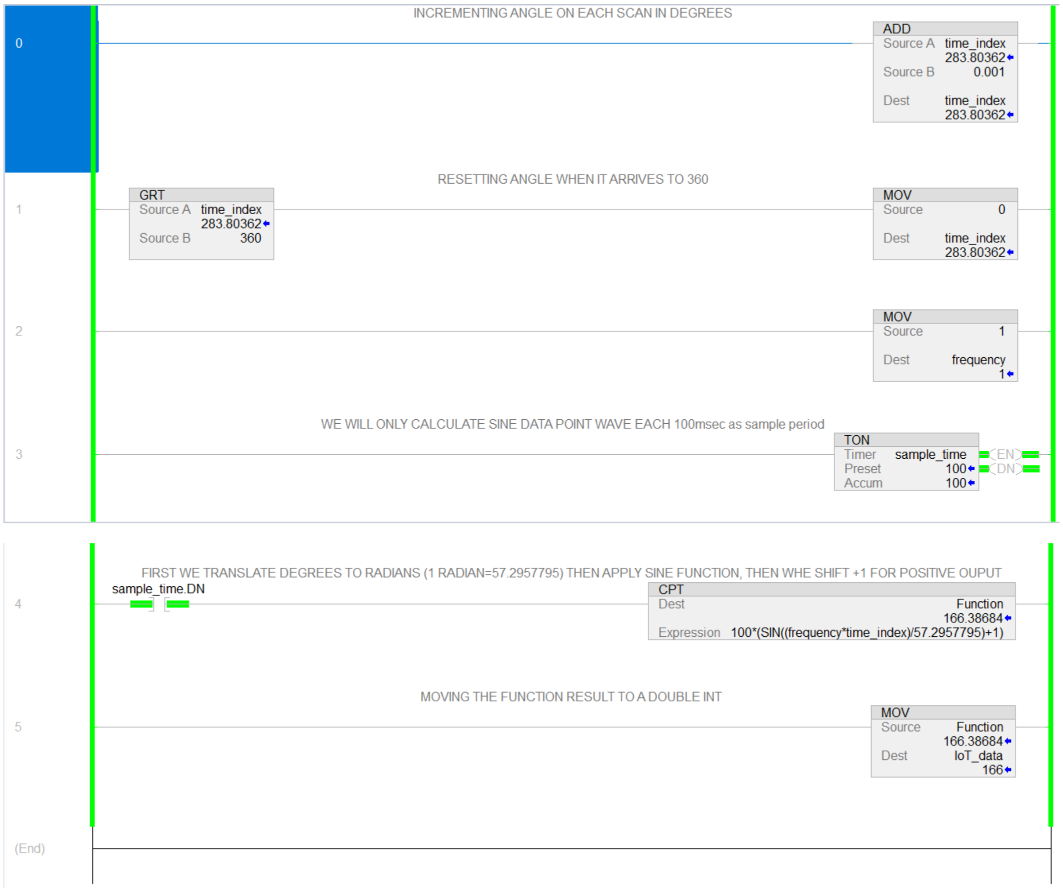
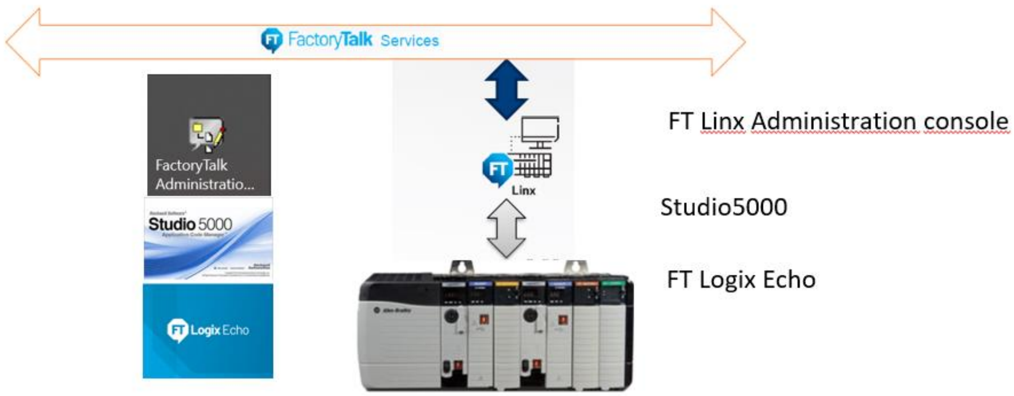


Figure 21 – Studio5000 screenshot with example program

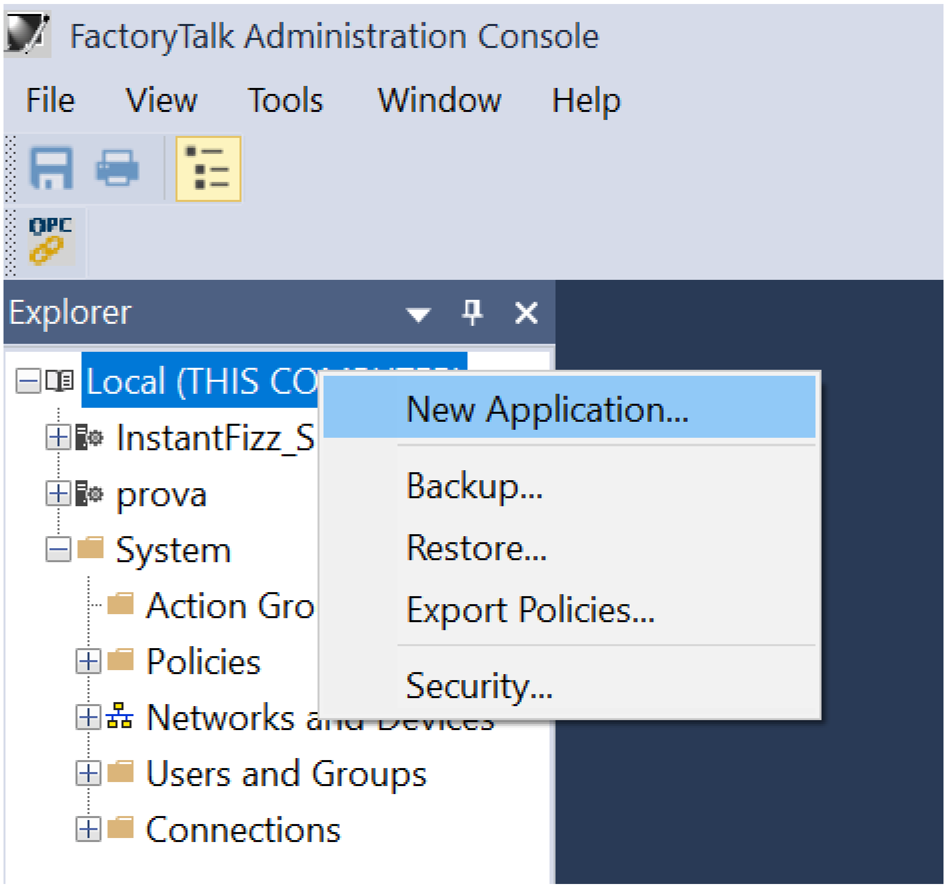
Next step we have to settle link the PLC to Factory Talk Services:



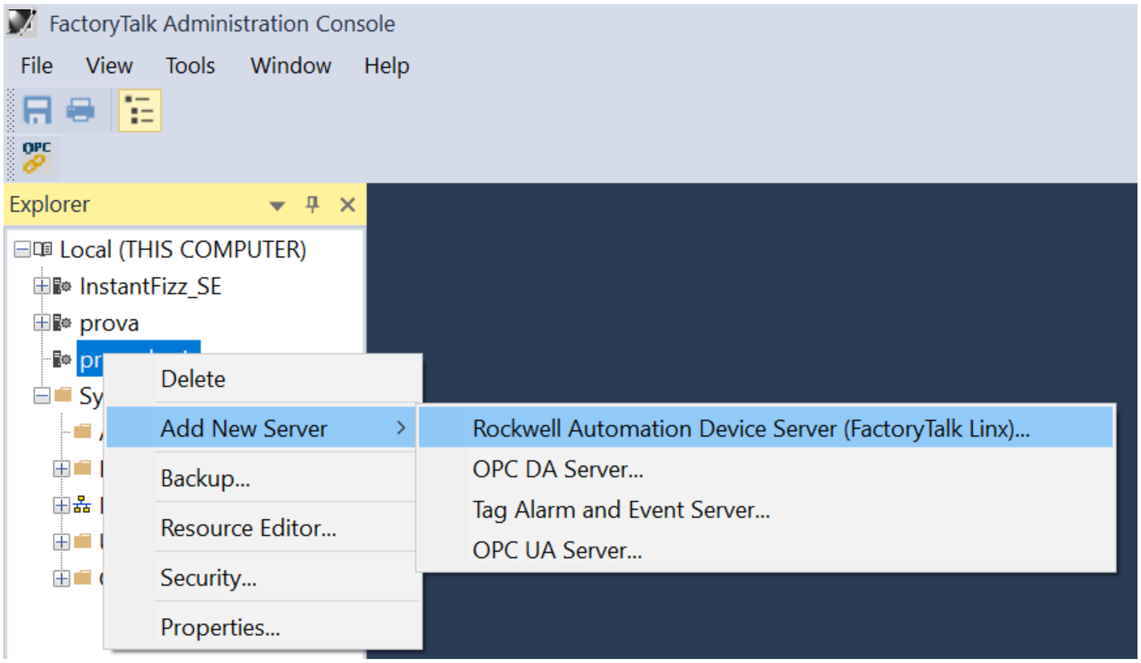
Now we open FT Linx Administration console

And take the local FT Directory.

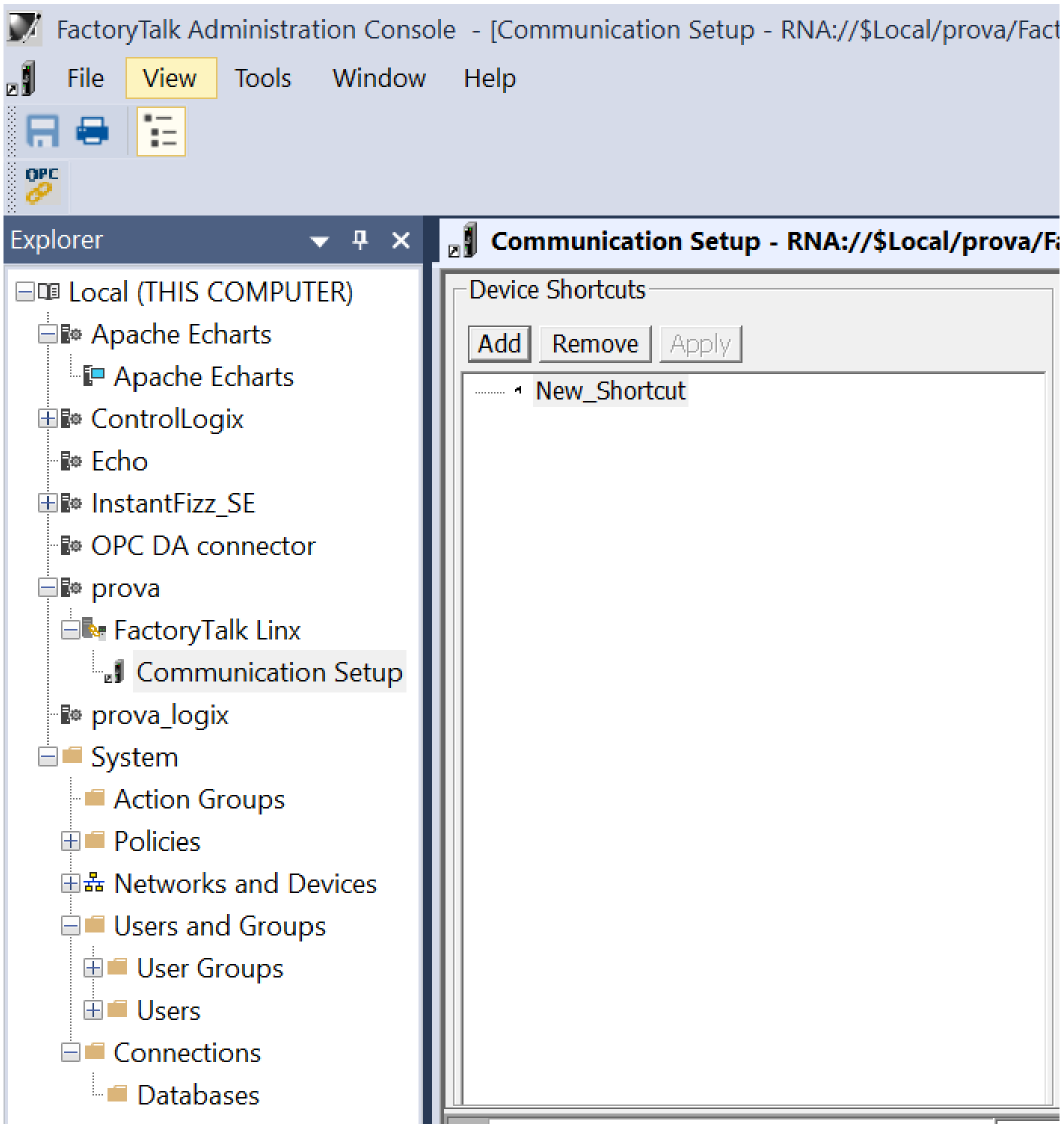
We add a new application



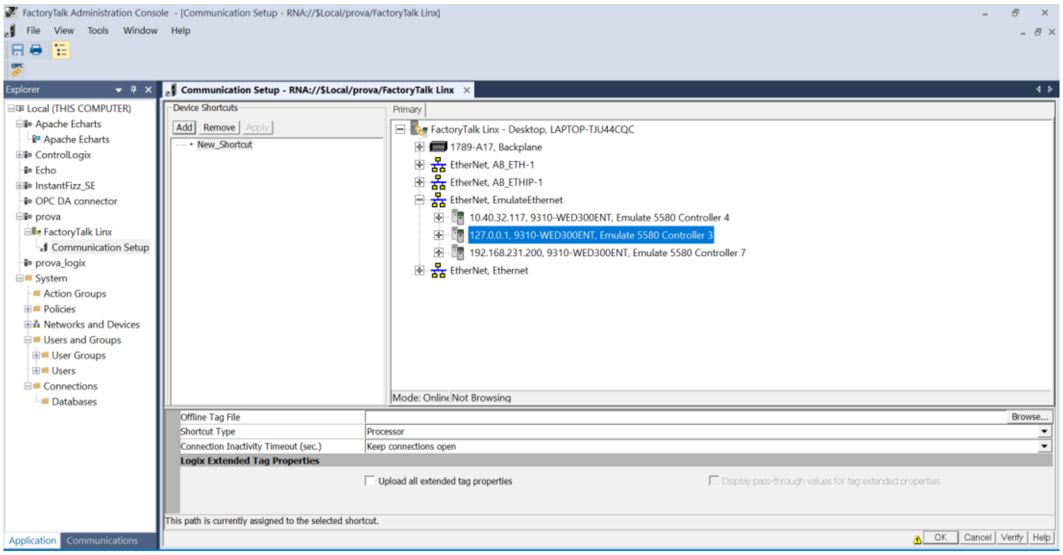
We add a new server



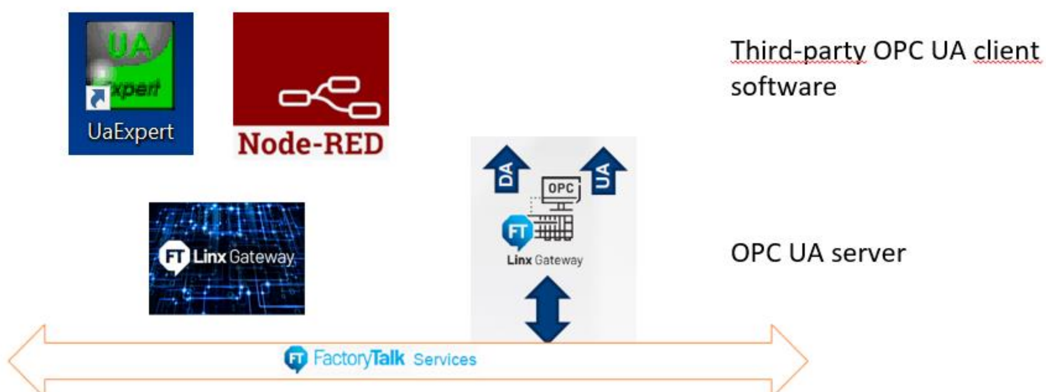
Double click on communications setup



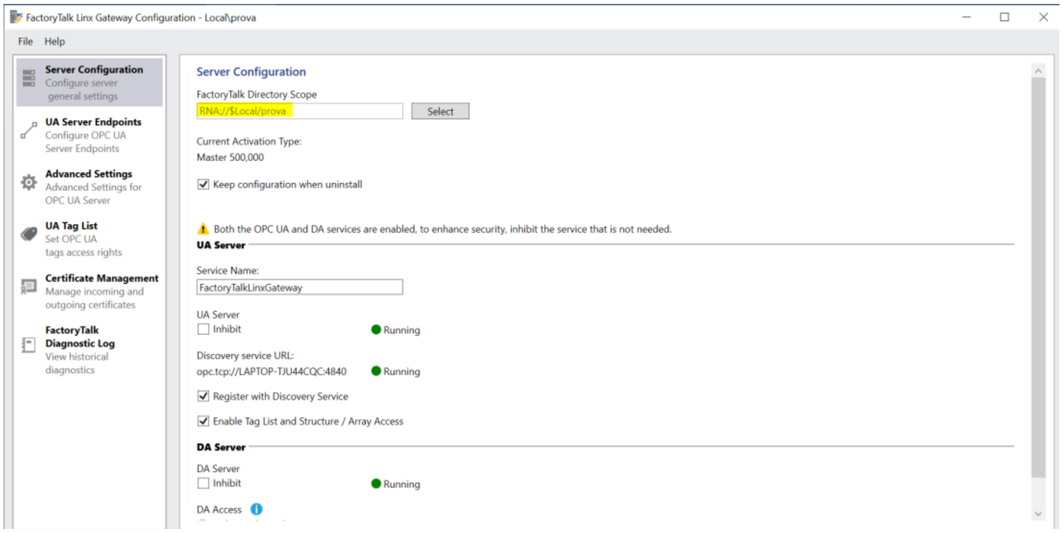
Add a new shortcut, click on Apply and OK



Now once we have the data on Factory Talk Services, we settle the OPC UA server: Factory Talk Linx Gateway:



Let’s open Factory Talk Linx Gateway Configuration and point to the last FT directory scope

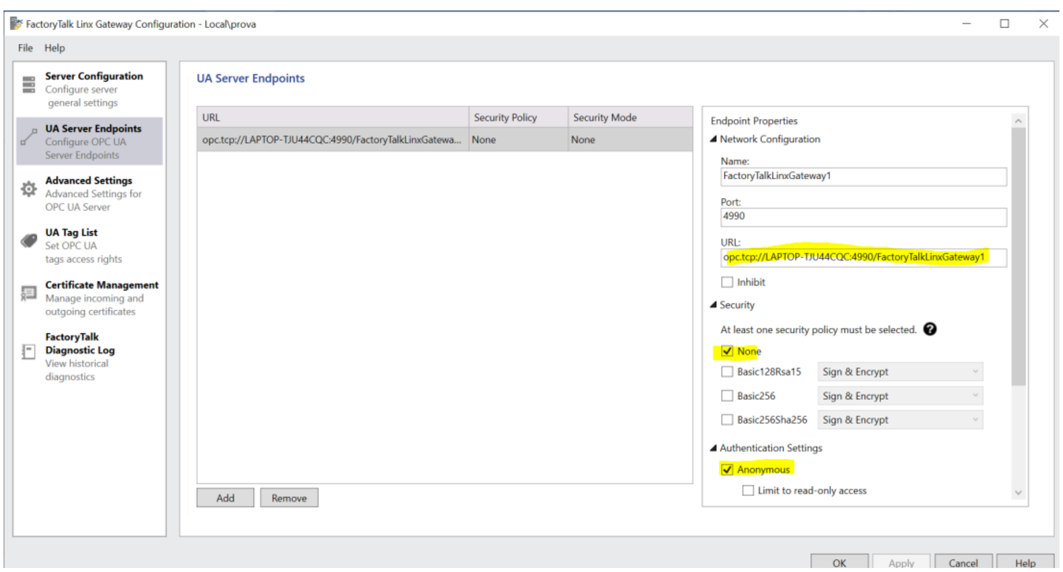


Let’s create an Endpoint

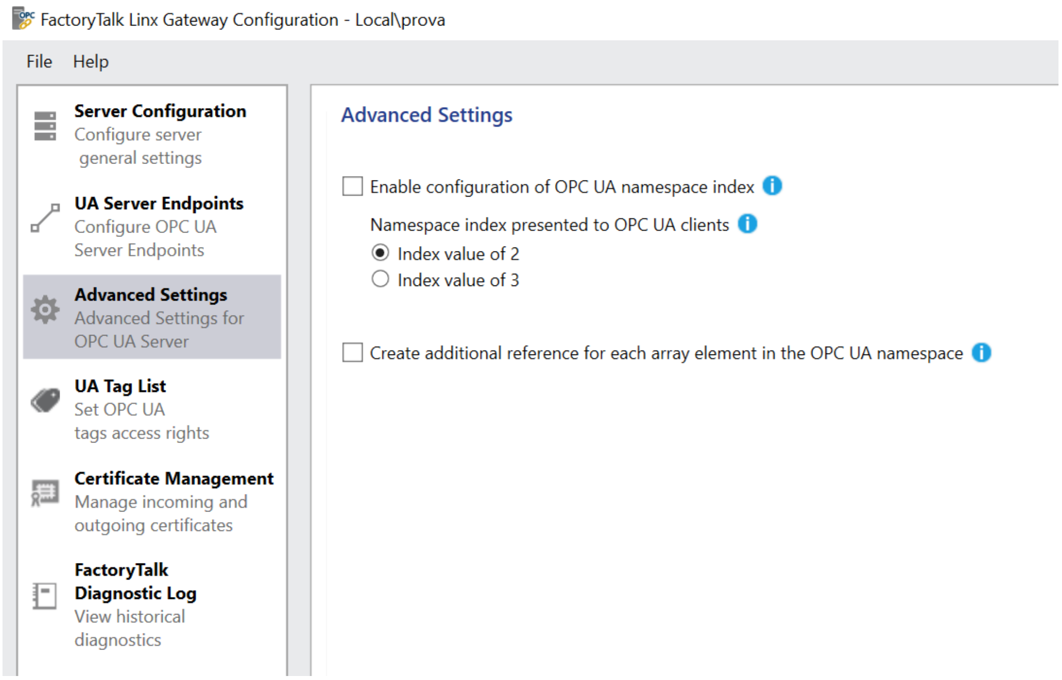
And copy this URL since we will use later on Node-RED:

opc.tcp://LAPTOP-TJU44CQC:4990/FactoryTalkLinxGateway1

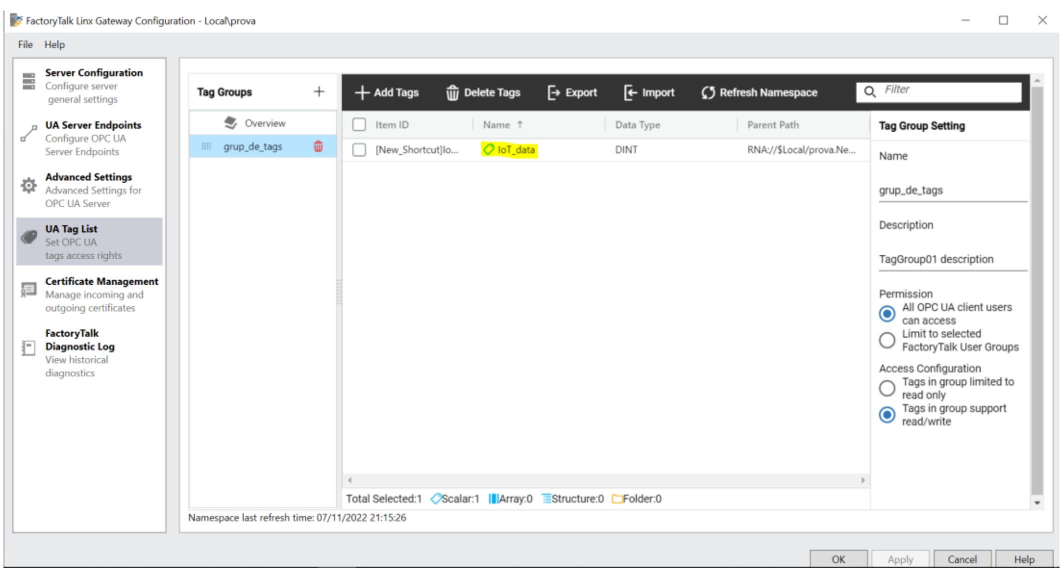
Use No security and anonymous for this use case



We can use defaults



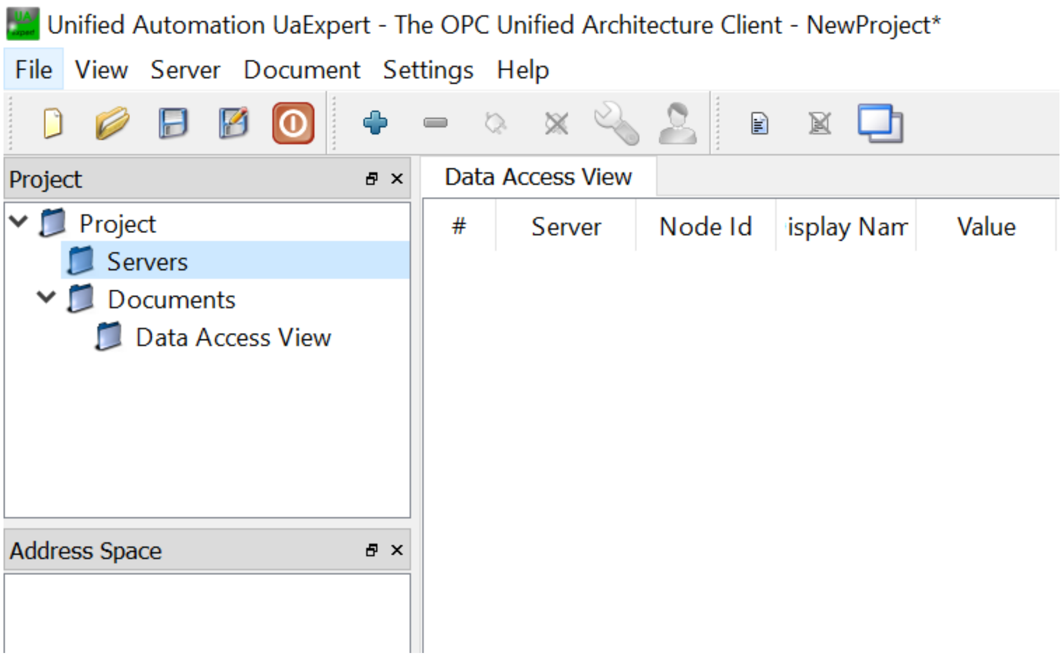
Add the desired tag



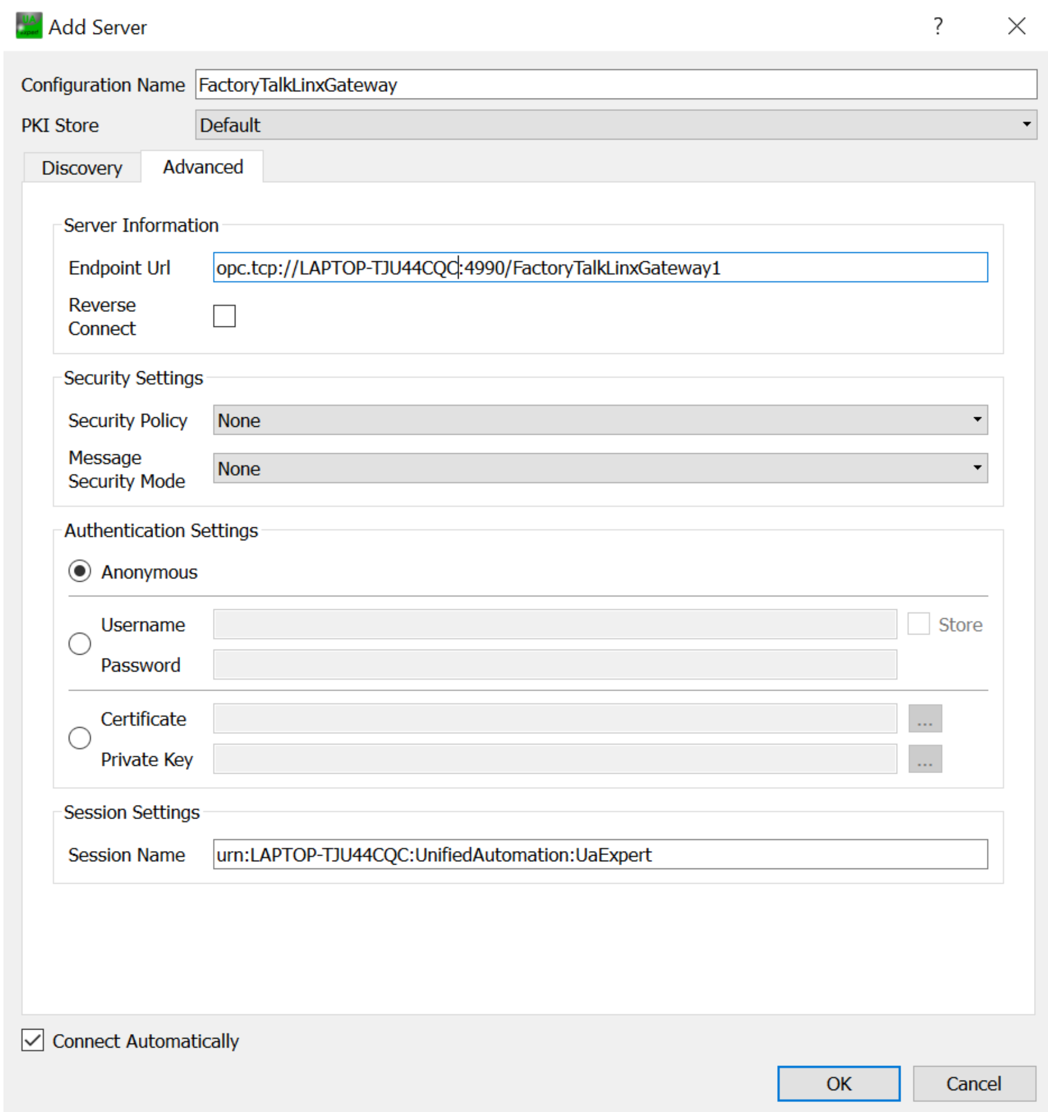
Click on Apply and OK

We have the OPC server ready and running.

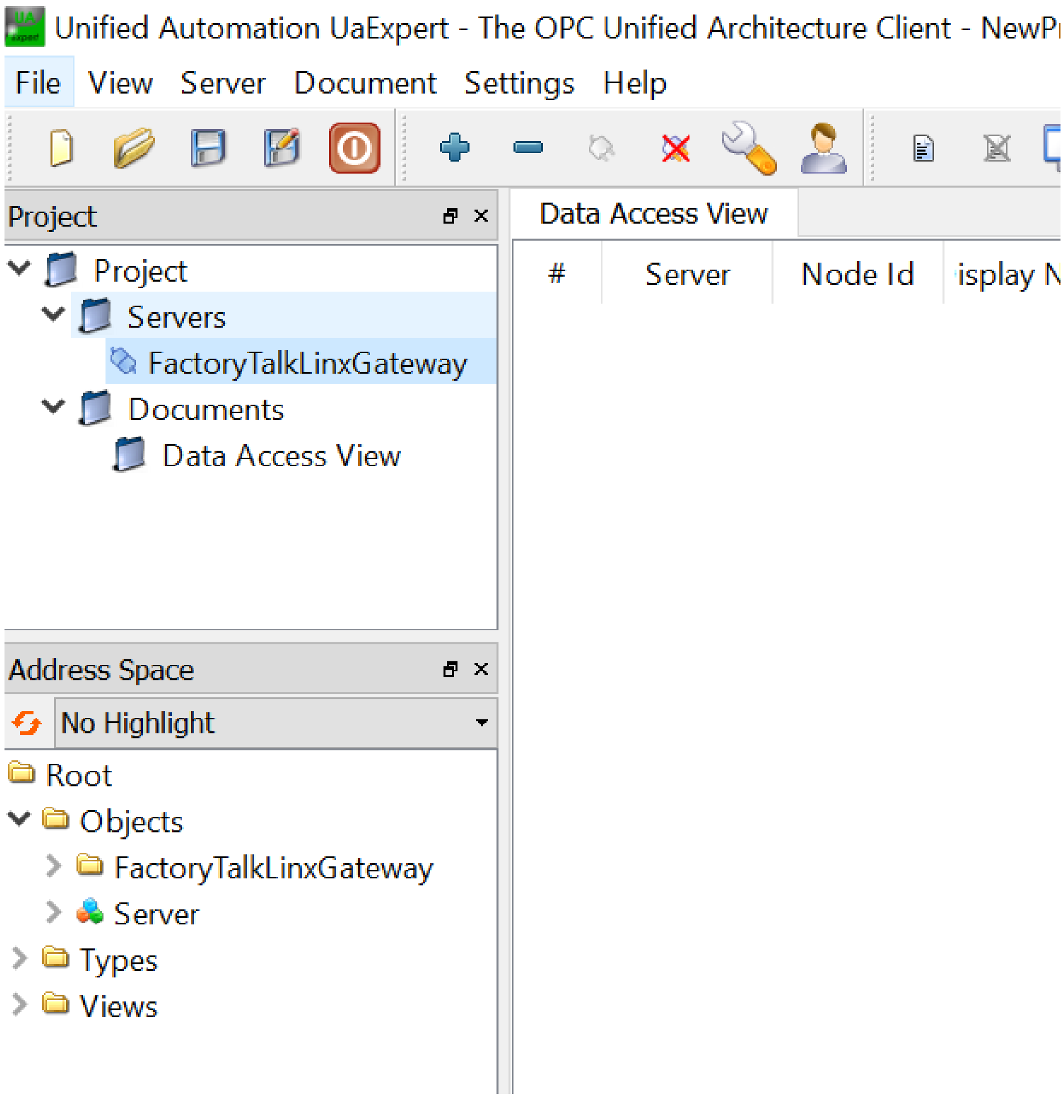
Let's go to a OPC client like UA Expert



Add a new server and use the created URL from server:

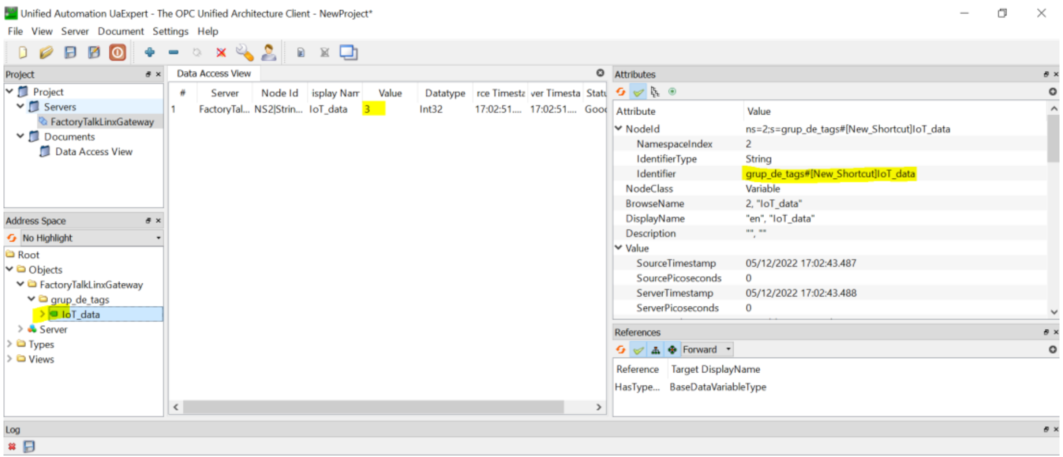


It gets automatically connection



Let's open the variables and copy the identifier value since we will use it later in Node-RED

grup_de_tags#[New_Shortcut]IoT_data



And then we go to Node-RED and install this node:

node-red-contrib-opcua

(install these nodes on your palette with “Manage Palette”)

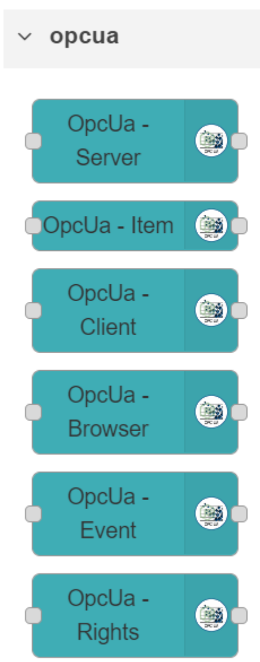


Figure 22 – OPC UA nodes

You have to use a not so late version on Node.js and node red, otherwise you would not be able to install the ethip nodes.

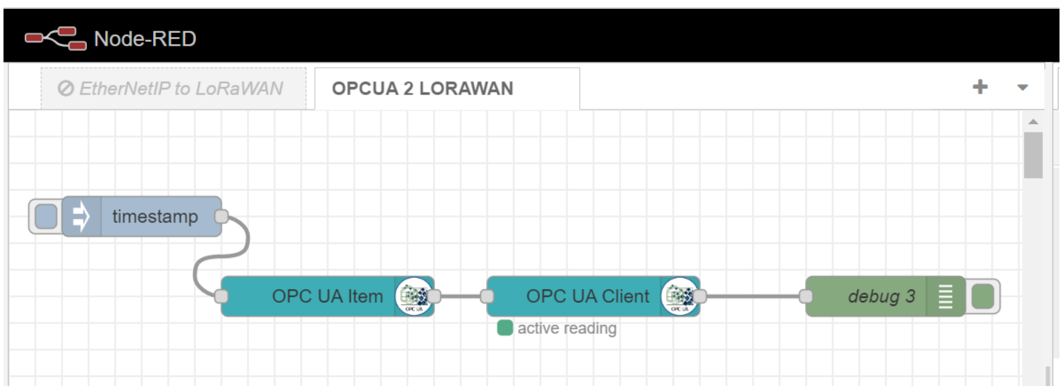
I have used

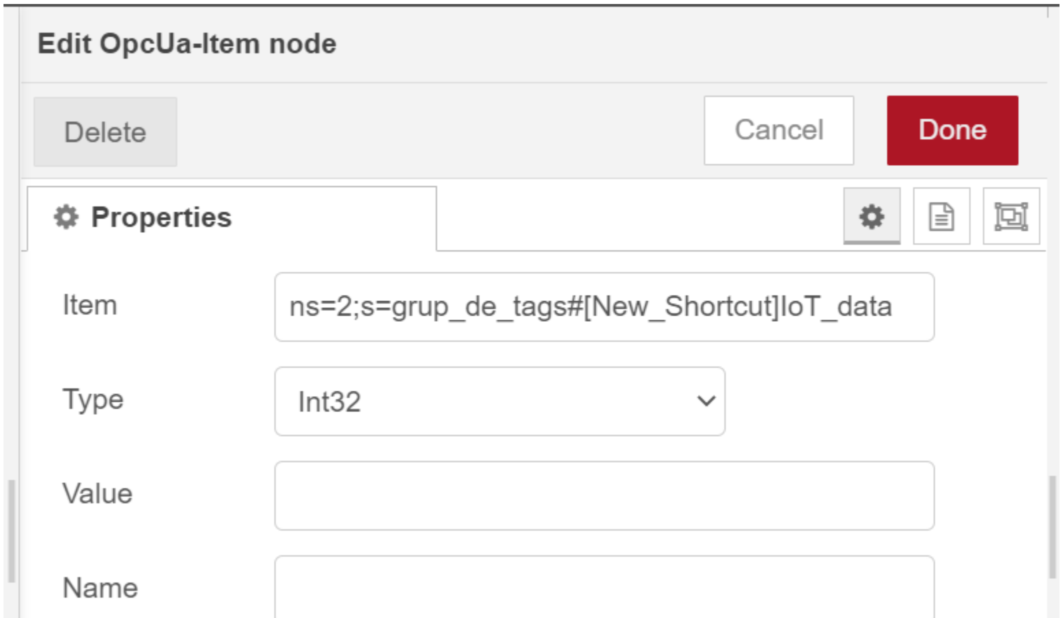
```
pi@raspberrypi:~ $ node -v
v16.16.0
pi@raspberrypi:~ $ npm -v
8.11.0
```

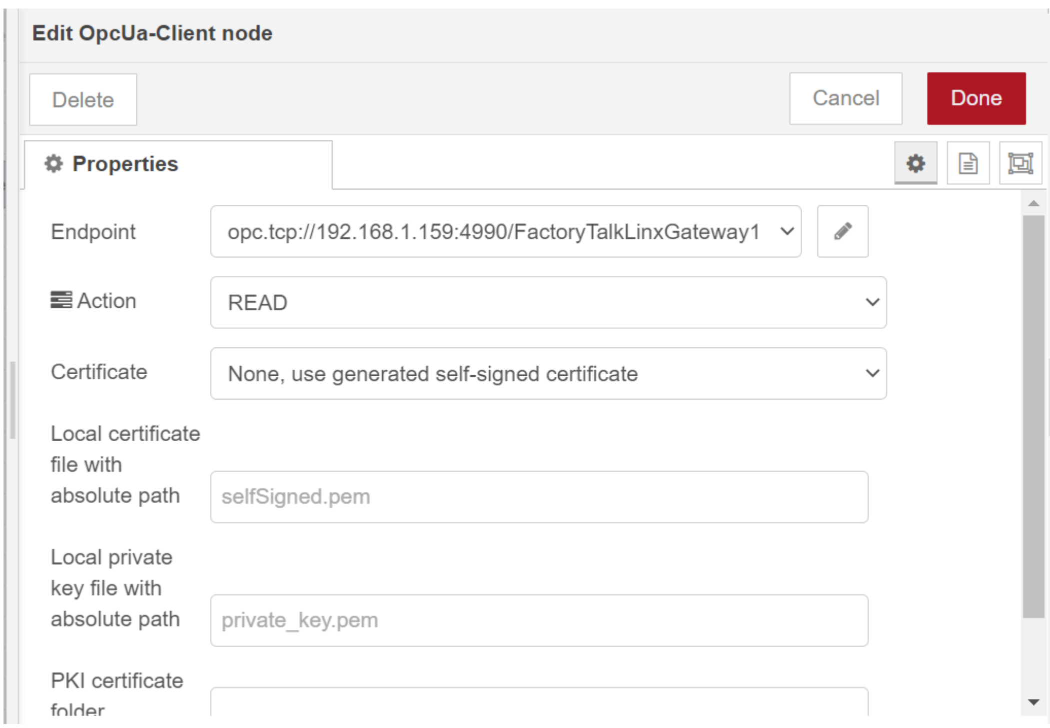
And node-red version 3.0.2

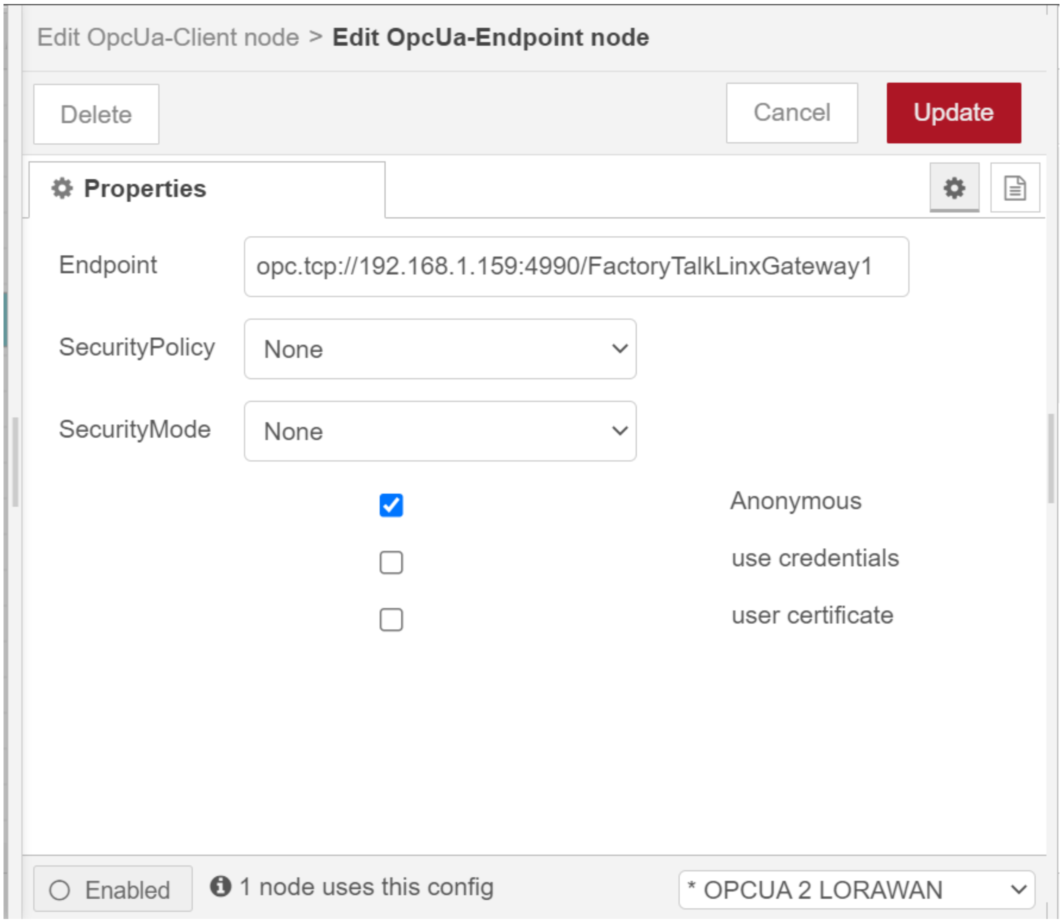
```
v3.0.2
```

Prepare the following flow for Node-RED as OPC UA server.

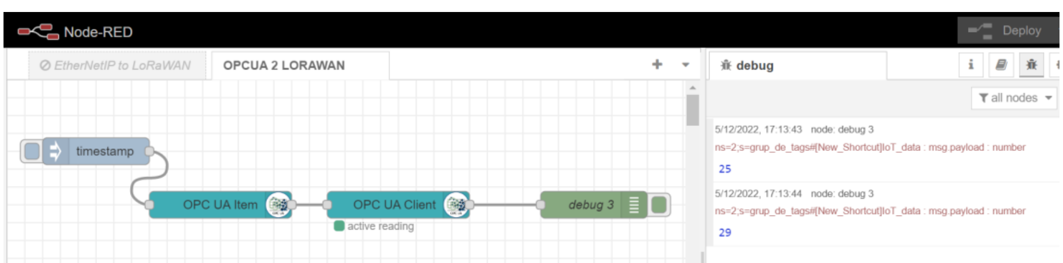


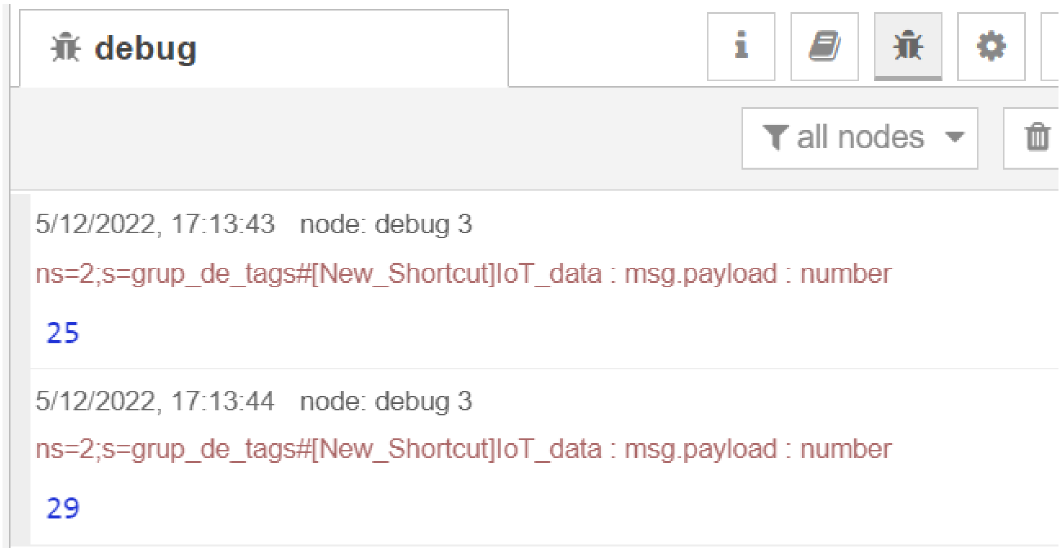

Edit OpcUa-Item node
Delete
Cancel
Done
Properties
Item
ns=2;s=grup_de_tags#[New_Shortcut]IoT_data
Type
Int32
Value
Name


Edit OpcUa-Client node
Delete
Cancel
Done
Properties
Endpoint
opc.tcp://192.168.1.159:4990/FactoryTalkLinxGateway1
Action
READ
Certificate
None, use generated self-signed certificate
Local certificate file with absolute path
selfSigned.pem
Local private key file with absolute path
private_key.pem
PKI certificate folder

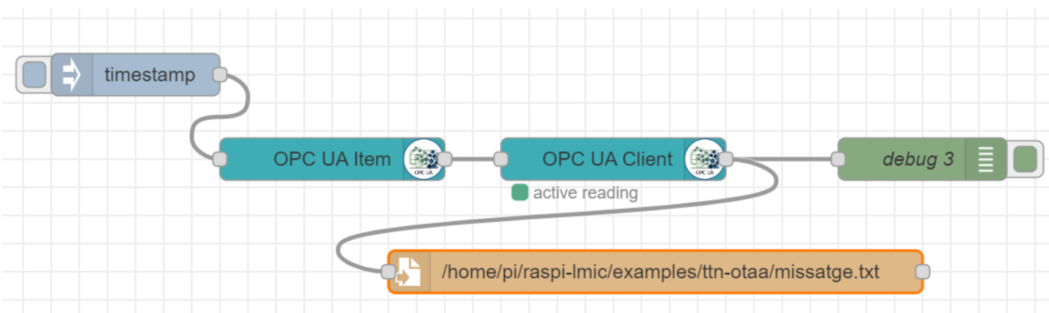


So each time we click on inject node, we see the data from OPC UA server

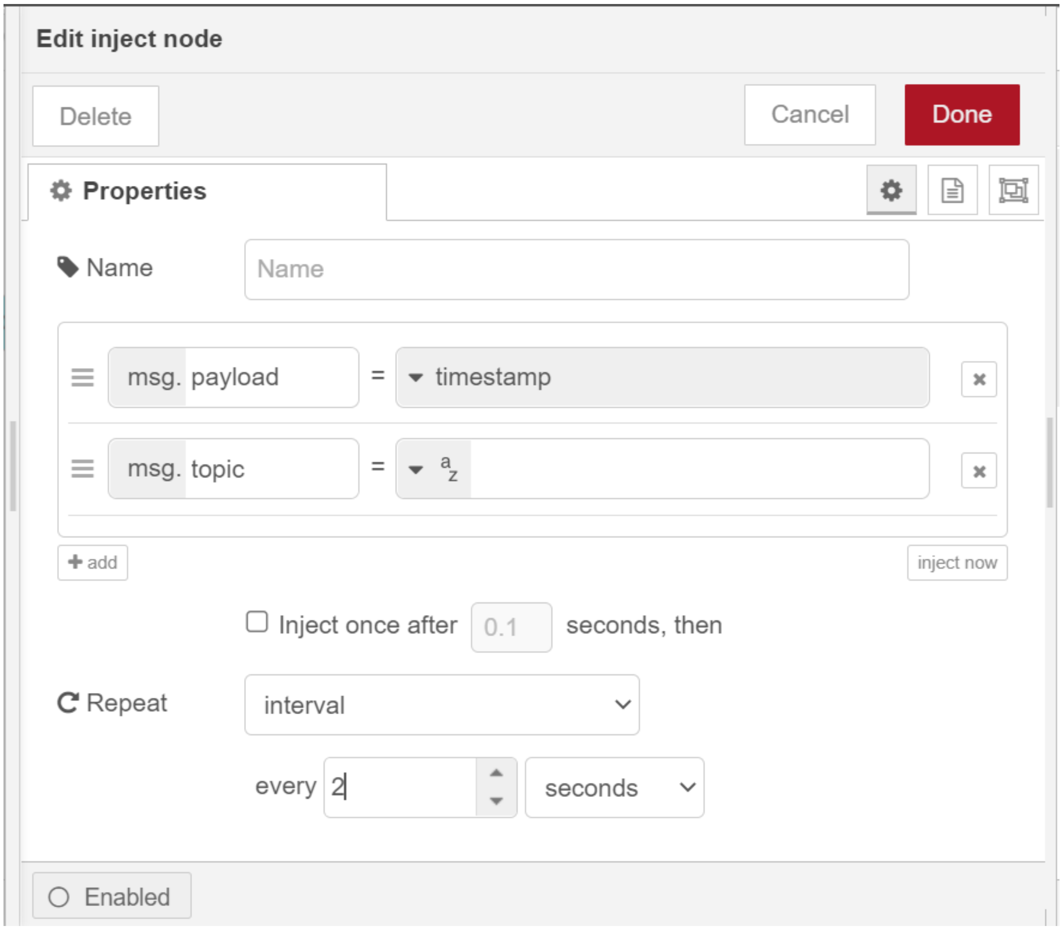


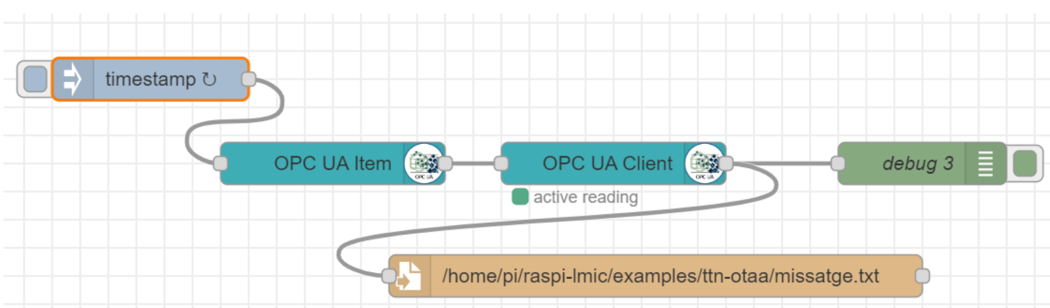


Now we have to write these values on the file we have tested before



But let's make this each two seconds





Now let’s take a look at the file contents

```
pi@raspberrypi:~ $ cd /home/pi/raspi-lmic/examples/ttn-otaa
pi@raspberrypi:~/raspi-lmic/examples/ttn-otaa $ cat missatge.txt
21
pi@raspberrypi:~/raspi-lmic/examples/ttn-otaa $ cat missatge.txt
4
pi@raspberrypi:~/raspi-lmic/examples/ttn-otaa $
```

You have everything ready for the final “Proof Of Concept” test.

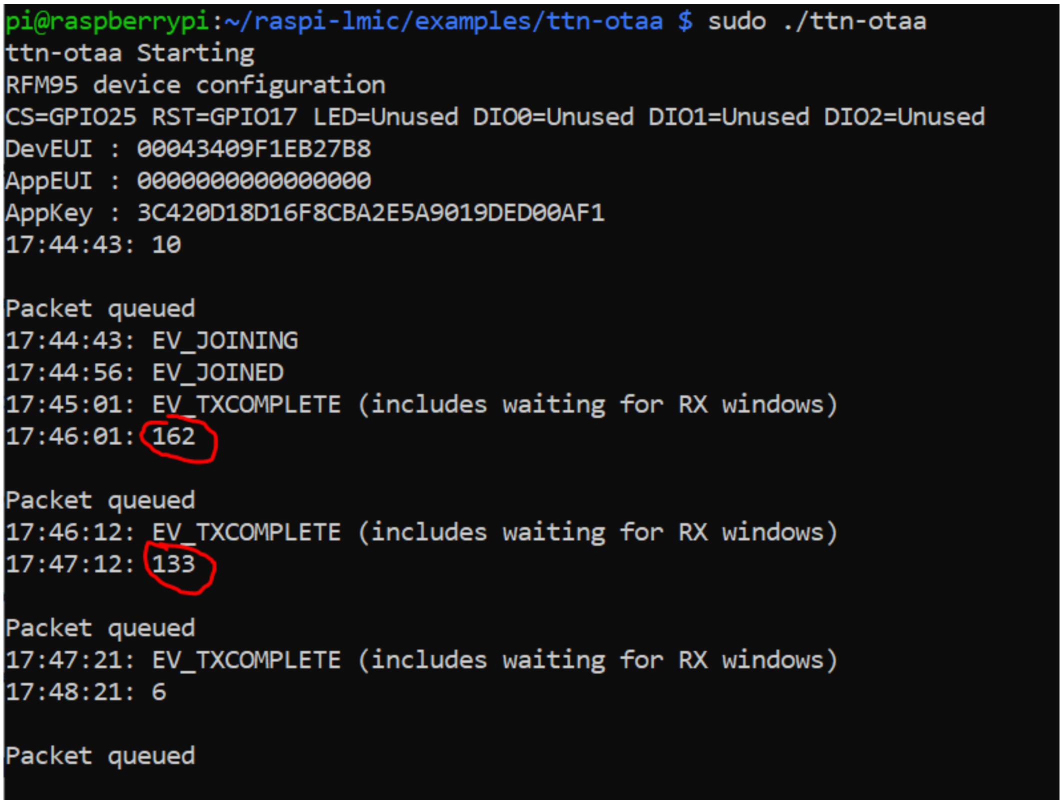
On next section you will be able to get data from a Ethernet/IP device, store it on a file, and send it thru LoRaWAN

Reading data from an OPC UA server and sending the data thru LoRaWAN

Let’s perform the final test: Open a terminal thru ssh to the converter, to see how the values are sent to LoRaWAN network. Use the IP address of your converter.

C:\Users\Risoul>ssh pi@raspberry.local

```
pi@raspberrypi:~/raspi-lmic/examples/ttn-otaa $ sudo ./ttn-otaa
ttn-otaa Starting
RFM95 device configuration
CS=GPIO25 RST=GPIO17 LED=Unused DIO0=Unused DIO1=Unused DIO2=Unused
DevEUI : 00043409F1EB27B8
AppEUI : 0000000000000000
AppKey : 3C420D18D16F8CBA2E5A9019DED00AF1
17:44:43: 10

Packet queued
17:44:43: EV_JOINING
```

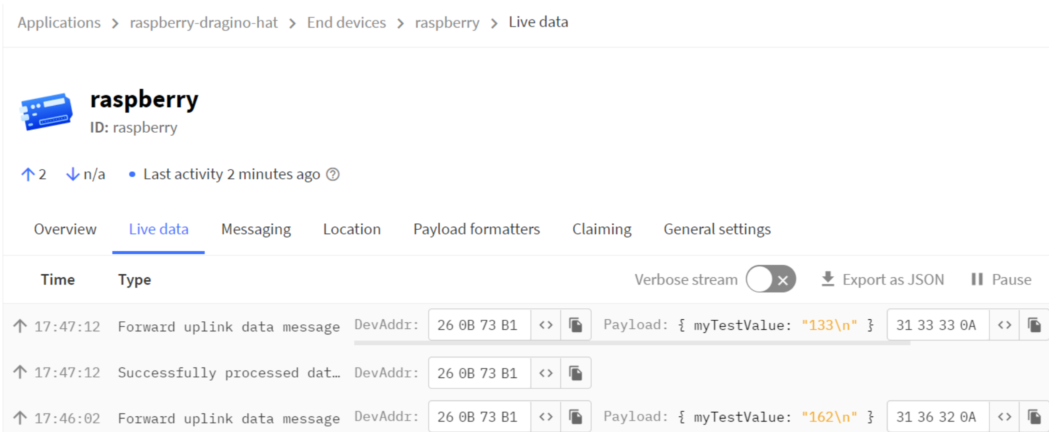


```
pi@raspberrypi:~/raspi-lmic/examples/ttn-otaa $ sudo ./ttn-otaa
ttn-otaa Starting
RFM95 device configuration
CS=GPIO25 RST=GPIO17 LED=Unused DIO0=Unused DIO1=Unused DIO2=Unused
DevEUI : 00043409F1EB27B8
AppEUI : 0000000000000000
AppKey : 3C420D18D16F8CBA2E5A9019DED00AF1
17:44:43: 10

Packet queued
17:44:43: EV_JOINING
17:44:56: EV_JOINED
17:45:01: EV_TXCOMPLETE (includes waiting for RX windows)
17:46:01: 162

Packet queued
17:46:12: EV_TXCOMPLETE (includes waiting for RX windows)
17:47:12: 133

Packet queued
17:47:21: EV_TXCOMPLETE (includes waiting for RX windows)
17:48:21: 6

Packet queued
```

Now we can go to the LoRaWAN network server and take a look at the uplink messages

It Works!

We get the value on the TTN console!



So now our raspberry is Reading from PLC, thru OPC UA with corresponding Tag value

You can find the complete Node-RED flow on following link:

https://github.com/xavierflorensa/OPC-UA-to-LoRAWAN/blob/main/OPCUA%20to%20LoRaWAN%20node-red%20flow.json

I have to congratulate you again, since you have managed to build your first linux based LoRaWAN node. It is not a low power node, so normally it will have to work plugged to the mains supply, but the Ethernet/IP devices which you normally will take the data from are also plugged to mains supply.

Summary

In this Document you have learned how to create a customized LoRaWAN node and you have got the chance to know the Raspi-LMIC library. You are now able to get data from an OPC UA server on Node-RED and to pass the data between software applications by using a file. You have gained knowledge on how to modify an application library in C++ language.