Manual del desarrollador

1.6.1

Envira Sostenible S.A.

Nanoenvi IAQ

envira

Tabla de contenidos

Lista de figuras

Lista de tablas

Capítulo 1. Control de revisiones

Tabla 1.1. Control de revisiones

Versión	Cambios
V1.6.1	Creada guía de integración Nanoenvi IAQ LoRa

Capítulo 2. Alcance del documento

El presente documento detalla aspectos de integración de Nanoenvi IAQ™ con red LoRaWAN y/o plataforma IoT, protocolo de mensajes *downlink* e información sobre la deserialización de las medidas.

Capítulo 3. Introducción

Nanoenvi IAQ™ cuenta con una variante con comunicaciones LoRaWAN. Esto permite instalar el dispositivo en edificios con cobertura de red LoRaWAN, donde es necesario medir la calidad del aire en tiempo real con independencia de redes de uso general como puedan ser redes WiFi o sin infraestructura de redes cableadas como pueda ser el caso de MODBUS.

Para funcionar, Nanoenvi IAQ™ necesita **instalación**: La instalación del dispositivo consiste en configurar la conexión y colocarlo en la ubicación en la que realizará las medidas. Requiere que tanto Network Server como Nanoenvi IAQ™ tengan la misma configuración. Para más detalles sobre la configuración relativa al Network Server utilizado, consulte la documentación del Network Server o acuda a su operador de red LoRaWAN. Para más detalles sobre el proceso de instalación, configuración y operación del dispositivo consulte el manual de usuario de Nanoenvi IAQ™.

Una ver realizada la instalación, durante su funcionamiento normal, la variante LoRaWAN de Nanoenvi IAQ™ toma datos de la calidad del aire interior en tiempo real y los envía usando la red LoRaWAN configurada. Las variables medidas por el dispositivo se serializan en formato protobuf.

Nanoenvi IAQ™ también soporta mensajes RPC (*remote procedure calls*) enviadas a través de mensajes *downlink* para editar su configuración y conocer su estado. El formato de serialización, tanto de mensajes *downlink* como *uplink* es protobuf (`https://developers.google.com/protocol-buffers`) .

Para la explotación de los datos y/o envío de mensajes remotos se requiere realizar la integración del dispositivo: la integración consiste en desplegar los servicios necesarios o configurar Network Server para que las medidas de los sensores, recibidas en el Network Server puedan ser usadas por otro sistema. Por ejemplo, si las medidas de los sensores se quieren mostrar en una plataforma de visualización, los mensajes recibidos en Network Server se deben deserializar, convertir al formato esperado por la plataforma de visualización y se deben reenviar desde el Network Server a la plataforma.

Este manual documenta toda la información necesaria para integrar Nanoenvi IAQ™.

Capítulo 4. Requerimientos de integración Nanoenvi IAQ™ LoRaWAN

A continuación se detallan las acciones necesarias para integrar Nanoenvi IAQ™:

4.1. Explotación de datos de sensores

Para la explotación de los datos tomados y enviados por el dispositivo se requiere:

- Un mecanismo que deserialice el *payload* del mensaje enviado por el dispositivo. El *payload* está serializado con protobuf.
- Un mecanismo que reenvíe los datos de Netowrk Server al sistema en el que se quieran explotar.

4.2. Actuación sobre el dispositivo mediante RPCs

La actuación sobre el dispositivo mediante RPCs es opcional. Requiere:

- La implementación de la serialización y composición de RPCs .
- Un mecanismo que implemente la lógica de envío de una RPC y recepción de la respuesta compatible con el modelo de comunicaciones RPC: se debe esperar a recibir un mensaje con medidas de sensores y una vez recibido enviar la RPC. Posteriormente se debe esperar a la respuesta de la RPC que tardará unos segundos en ser recibida.
- La implementación de la deserialización de respuestas a RPCs.

Capítulo 5. Formato de mensajes de medidas de sensores

5.1. Formato de mensajes de medidas de sensores

Los valores de medidas tomados por los sensores de Nanoenvi IAQ™ son serializados con protobuf en un mensaje con la siguiente estructura (descrita en lenguaje proto3):

```
syntax = "proto3"; //Proto3 syntax

/**
 * Sensor measurements message format
 */
message SensorMeasurements {
    uint32 msg_ver = 1;  /// Message version to track changes. This number is included
                         // along all the SensorMeasurements versions and increased
                         // each time message format changes.
    float  co2     = 2;  /// CO2 measurement value in ppm units
    float  voc     = 3;  /// VOC measurement value in ppm units
    float  co      = 4;  /// CO measurement value in ppm units
    float  noise   = 5;  /// Noise level percentage
    float  pm10    = 6;  /// PM10 concentration in ug/m3
    float  pm2_5   = 7;  /// PM2.5 concentration in ug/m3
    float  temp    = 8;  /// Temperature in Celsius degrees
    float  hum     = 9;  /// Realtive humidity in % units
    float  prb     = 10; /// Barometric pressure in HPa units
    float  pm1     = 11; /// PM1 concentration in ug/m3
    float  pm4     = 12; /// PM4 concentration in ug/m3
    float  iaqi    = 13; /// Indoor air quality index
    float  tci     = 14; /// Thermal comfort index
    float  eiaqi   = 15; /// Environmental indoor air quality index
}
```

Los valores de *co* y *noise(*ruido*)* son opcionales en función de la variante del dispositivo: por restricciones del hardware, Nanoenvi IAQ™ sólo puede tener uno de los dos sensores (CO o ruido) .

El mensaje con las medidas es enviado cada (aproximadamente) 70 segundos y (al igual que cualquier otro tipo de mensaje LoRAWAN *uplink*) será recibido en el Network Server. Desde el Network Server el mensaje se puede reenviar y/o deserializar. Para ello consulte la documentación de su Network Server.

Capítulo 6. Mensajes de configuración

6.1. RPC

Nanoenvi IAQ™ implementa *remote procedure calls* (RPC). Las *remote procedure calls* son mensajes *downlink* que Nanoenvi IAQ™ puede recibir, interpretar y ejecutar. Los mensajes implementados permiten cambiar la configuración, leer configuración, calibrar el dispositivo y actuar sobre él (por ejemplo resetearlo).

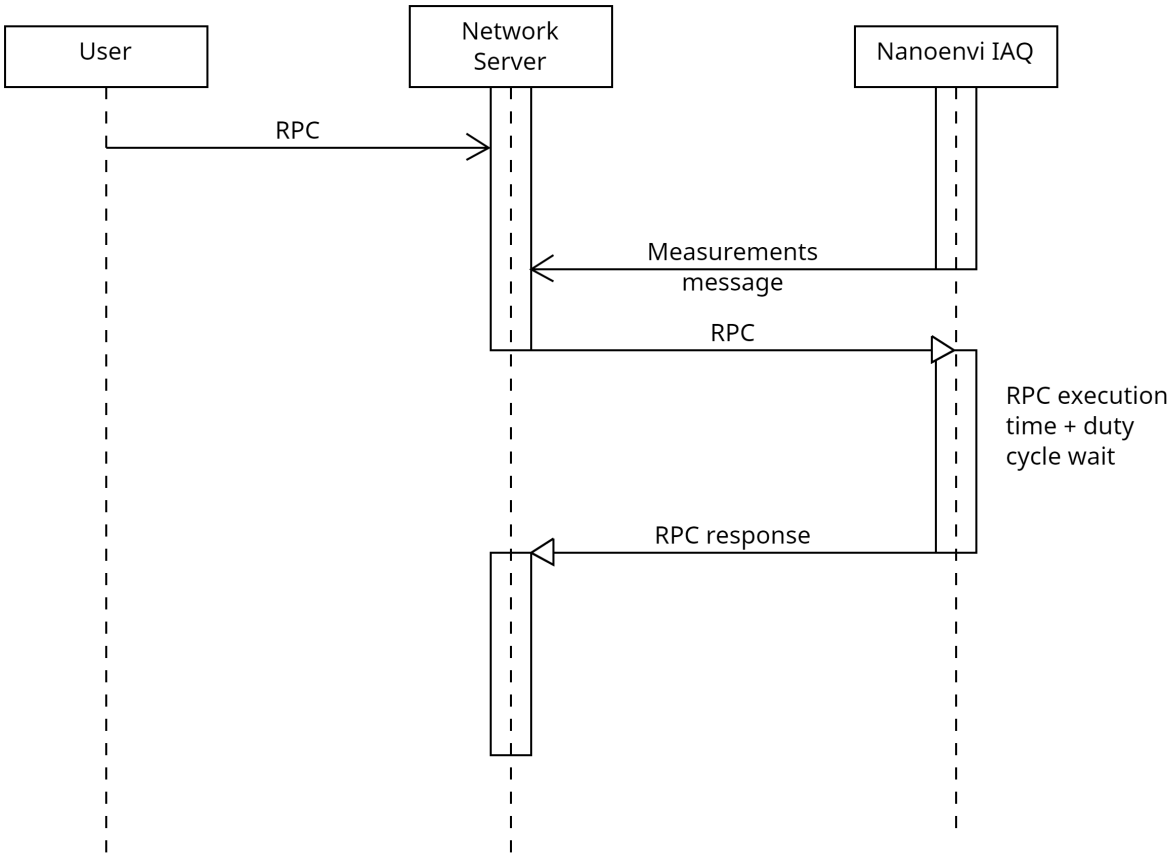
6.1.1. Modelo de comunicaciones RPC

Una vez conectado a la red LoRaWAN, Nanoenvi IAQ™ realiza medidas de los sensores y envía los resultados de las medidas. Al tratarse de un dispositivo LoRaWAN de clase A, justo tras realizar un envío de medidas, se abre una ventana de escucha de mensajes *downlink* . En caso de recibir un mensaje *downlink,* Nanoenvi IAQ™ interpreta el payload del mensaje y si contiene una RPC soportada, con argumentos y formato válido la ejecuta: las RPCs pueden cambiar la configuración, pedir información o actuar sobre el dispositivo. Tras la ejecución de la RPC, Nanoenvi IAQ™ genera una respuesta (con el resultado de la ejecución de la RPC), espera el tiempo correspondiente al *duty cycle* y envía dicha respuesta a través de la red LoRaWAN (esta respuesta será recibida por tanto en el Network Server).

Si el mensaje *downlink* contiene un payload con errores de formato, una RPC no soportada o se produce algún fallo relacionado con su interpretación o ejecución, el dispositivo esperará un tiempo correspondiente al *duty cycle* y enviará un mensaje de error.

La siguiente figura muestra de forma esquemática la secuencia de envío de mensajes entre Network Server y RPC y Nanoenvi IAQ™ cuando el usuario envía una RPC:

Figura 6.1. Modelo de comunicaciones RPC



Si bien es posible enviar mensajes consecutivos al dispositivo antes de que éste responda a la primera RPC enviada, esto no es recomendable y puede dar lugar a una respuesta inestable por parte del mismo. Por lo tanto, se recomienda respetar el modelo comunicaciones y para cada mensaje esperar a la respuesta antes de enviar la siguiente RPC.

Los mensajes RPC son procesados en orden de recepción y hay mensajes que causan un reset del dispositivo. Se debe tener en cuenta que tras un reset el dispositivo éste tardará un tiempo en reconectar a la red LoRaWAN.

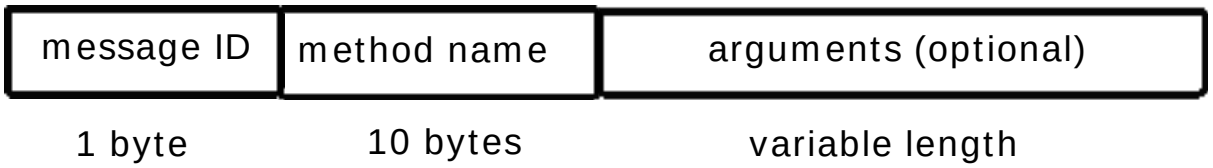
Para integrar Nanoenvi IAQ™ con un Network Server se debe tener en cuenta este modelo de comunicaciones. Si se quiere permitir el envío de RPCs al dispositivo se debe desplegar un mecanismo que permita enviar una RPC justo después de recibir un mensaje con medidas de sensores, que espere la respuesta RPC la decodifique y la reporte.

6.2. Formato de RPCs

6.2.1. Formato de RPCs enviados a Nanoenvi IAQ™

El payload de mensajes RPC en formato binario que espera recibir Nanoenvi IAQ™ deben tener la siguiente estructura:

Figura 6.2. formato mensaje RPC

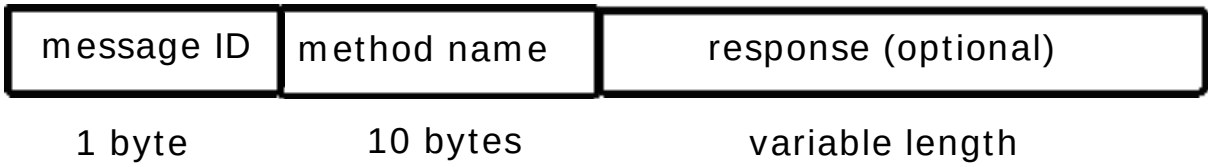


- ***message ID***: 1 byte que se corresponde con el ID del mensaje: un número entre 0 y 255 que identifica el mensaje. Su uso es orientativo para identificar los mensajes, en la práctica no es usado por el dispositivo.
- **method name**: nombre del método RPC. Debe ocupar 10 bytes. En caso de que la longitud del nombre del método sea menor de 10 bytes, los bytes restantes deben tener valor 0x01.
- **arguments**: Contiene los argumentos del método RPC serializados con Protobuf. Dado que algunos métodos no tienen argumentos, este campo es opcional.

6.2.2. Formato de respuestas RPCs enviados por Nanoenvi IAQ™

La respuesta a los mensajes RPC en formato binario generada por Nanoenvi IAQ™ tras recibir y procesar un mensaje RPC consiste en un número variable de bytes con el siguiente formato:

Figura 6.3. formato respuesta RPC



- **message ID**: 1 byte que se corresponde con el ID del mensaje al que responde el dispositivo.
- **method name**: nombre del método RPC. Debe ocupar 10 bytes. En caso de que la longitud del nombre del método sea menor de 10 bytes, los bytes restantes tendrán valor 0x01. Coincide con el nombre del mensaje recibido por Nanoenvi IAQ y procesado al que se responde.
- **response**: Valores devueltos en el mensaje como respuesta del método RPC. Su longitud es variable y están serializados con Protobuf.

6.2.2.1. respuesta de OK genérica

Para algunas RPC, se devuelve una respuesta de OK genérica que tiene los siguientes valores:

- **message ID**: byte con el mismo valor del mensaje RPC al que se responde.
- **method name**: 10 bytes con el mismo valor del nombre del método del mensaje RPC al que se responde.
- **response:** en el caso de la respuesta de OK genérica, este campo contiene la cadena de caracteres "ok" serializada con protobuf en un mensaje con la siguiente estructura (descrita en lenguaje proto3):

```
/**
 * Protobuf RPC genric OK response
 */
syntax = "proto3"; //Proto3 syntax
```

```
/**
 * RPC OK generic response
 */
message RpcOkArgs {
    bytes message =1; /// String to return OK response
}
```

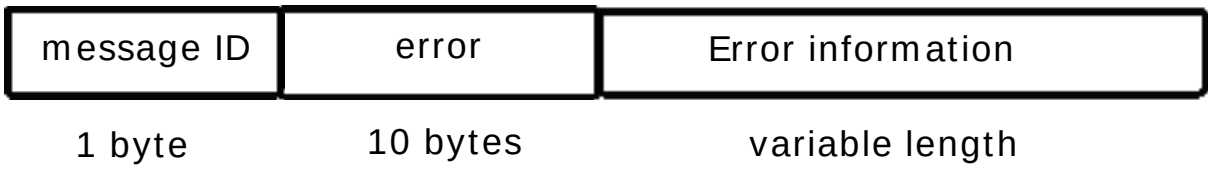
La respuesta sólo tiene un campo:

- **message:**siempre se espera que contenga la cadena de caracteres "ok"

6.2.3. Respuestas de error

En caso de error en la interpretación de un mensaje RPC Nanoenvi IAQ™ puede responder diferentes mensajes en función del error. Todos ellos tienen la misma estructura:

Figura 6.4. formato respuesta RPC con error



- **message ID**: 1 byte que se corresponde con el ID del mensaje que generó el error.
- **error**: 10 bytes que indican que se trata de un mensaje de error. El valor de los 10 bytes será: `0x65 0x72 0x72 0x6F 0x72 0x01 0x01 0x01 0x01 0x01`
- **error information**: Información sobre el tipo de error serializada en formato protobuf. Su longitud es variable y menor de 30 bytes. Contiene dos campos:
  - **code**: número entero de 32 bits con código de error.
  - **message**: cadena de caracteres con información del error.

  A continuación se muestra la estructura de un mensaje de error descrita en lenguaje proto3:

```
syntax = "proto3";

/**
 * RPC error response fields
 */
message RpcErrorArgs {
    int32 error_code = 1;  /// Error code
    bytes message =2; /// Error message (16 bytes max.)
}
```

La siguiente tabla recoge los posibles errores y su significado:

error_code	message	error
-32602	bad params	El mensaje recibido tiene parámetros incorrectos (fuera de rango).

-32601	method not found	El dispositivo no implementa el método RPC
-32700	parse error	El mensaje recibido tiene errores de serialización o su formato no es correcto
-32600	inval request	El mensaje recibido no se corresponde con una petición RPC válida
-32000	aborted	Al procesar y ejecutar la RPC se abortó el proceso
-32001	busy	Al ejecutarse la RPC se solicitó el uso de un recurso ocupado
-32002	timeout	Al ejecutarse la RPC se produjo un timeout
-32003	denied	Se denegó la ejecución de la RPC
-32004	not initalized	Al ejecutarse la RPC se solicitó un recurso del dispositivo que no ha sido inicializado
-32005	failure	Al ejecutarse la RPC se produjo un fallo
-32006	forbidden	La RPC solicita una operación prohibida
-32007	unknown	Al ejecutarse la RPC se produjo un error desconocido

6.3. Listado de RPCs públicas

A continuación se listan los métodos RPC públicos soportados por el dispositivo:

- **Co2Frc**: permite indicar al sensor de CO2 la concentración actual de CO2 (en ppm) para su calibración.
- **reset**: resetea el dispositivo.
- **SetOtaa**: edita la configuración de AppKey y AppEUI/JoinEUI.
- **FactoryCfg**: Resetea la configuración a valores de fábrica.

En caso de que AppKey y/o AppEUI/JoinEUI configurados en Network Server tengan valores distintos de los de por defecto, la invocación de este método hará que se pierda conexión con la red LoRaWAN

- **FwData**: consulta metadatos de la versión de firmware.
- **DevInfo**: consulta la fecha de fabricación y versión de hardware.

6.4. Documentación RPCs públicas

6.4.1. Co2Frc

Este método permite indicar una concentración de referencia para el sensor de CO2 con el fin de calibrarlo. El sensor de CO2 del dispositivo puede dar medidas erróneas si se expone a cantidades muy altas de CO2 de forma continuada por un tiempo superior a 7 días o si el dispositivo sufre algún impacto.

Para recalibrar el sensor se puede exponer a una cantidad de CO2 conocida o tomar la medida de un medidor de referencia calibrado e indicar la concentración de CO2 real mediante el método RPC *Co2Frc*. Si no se conoce la concentración de CO2 (bien porque no se tiene la capacidad de generar una concentración de CO2 de referencia o porque no se dispone de un medidor de referencia), se puede instalar el dispositivo en el exterior durante 5 minutos y suponer que la concentración real de CO2 es de 400 ppm.

6.4.1.1. Mensaje *downlink*

Los campos del mensaje *downlink* deben tener los siguientes valores:

- **Method name:** "Co2Frc". En hexadecimal: `0x43 0x6F 0x32 0x46 0x72 0x63 0x01 0x01 0x01 0x01`
- **Arguments:** Los argumentos de la RPC deben estar serializados en formato protobuf. A continuación se indica la estructura de los argumentos descrita en lenguage proto3:

```
/**
 * SCD30 forced recalibration arguments structure definition
 */
syntax = "proto3"; //Proto3 syntax

message Scd30RpcFrcArgs
{
    uint32 c_ref_ppm = 1 ; // CO2 reference concentration.
                           // It should be the actual concentration that the
                           // sensor should be reading
}
```

Los argumentos sólo contienen un campo:

- **c_ref_ppm**: concentración real de CO2 en ppm. Su valor debe estar entre 400 y 2000 ppm.

6.4.1.2. Respuesta

En caso de recepción y ejecución sin errores del método, se devuelve una respuesta de ok genérica tal y como se especifica en la sección correspondiente del presente manual. En caso de encontrarse algún error, la respuesta de error será una respuesta de error con el formato documentado en el apartado correspondiente del presente manual.

6.4.2. reset

Este método resetea de forma remota el dispositivo. Tras recibir la RPC con este método, el dispositivo no se resetea hasta enviar el mensaje de respuesta al método.

6.4.2.1. Mensaje *downlink*

Los campos del mensaje *downlink* deben tener los siguientes valores:

- **Method name:** "reset". En hexadecimal: `0x72 0x65 0x73 0x65 0x74 0x01 0x01 0x01 0x01 0x01`
- **Arguments:** Este método no tiene argumentos

6.4.2.2. Respuesta

En caso de recepción y ejecución sin errores del método, se devuelve una respuesta de ok genérica tal y como se especifica en la sección correspondiente del presente manual. En caso de encontrarse algún error, la respuesta de error será una respuesta de error con el formato documentado en el apartado correspondiente del presente manual.

6.4.3. DevInfo

Esta RPC solicita versión de hardware e identificador del dispositivo.

6.4.3.1. Mensaje *downlink*

Los campos del mensaje *downlink* deben tener los siguientes valores:

- **Method name:** "Devinfo". En hexadecimal: `0x44 0x65 0x76 0x69 0x6E 0x66 0x6F 0x01 0x01 0x01 0x01`
- **Arguments:** Este método no tiene argumentos

6.4.3.2. Respuesta

En caso de recepción y ejecución sin errores del método, se devuelve la versión de hardware y el tipo de dispositivo en la respuesta, serializados en formato protobuf con la siguiente estructura:

```
syntax = "proto3";

message HwVer
{
    uint32 major  = 1; // Hardware major version
    uint32 minor  = 2; // Hardware minor version
    uint32 patch  = 3; // Patch identifier
}

message DeviceType
{
    bytes device_type  = 1; // Device reference name
}

message RpcGetDeviceInfoArgs
{
    HwVer      version     = 1;
    DeviceType device_type = 2;
}
```

Los metadatos tienen los siguientes campos:

- **HwVer**: Estructura de tres campos con números enteros (major, minor y patch) que identifican la versión del hardware del dispositivo.
- **DeviceType**: cadena de caracteres con codificación ASCII con el identificador del tipo de dispositivo (Nanoenvi IAQ).

En caso de encontrarse algún error, la respuesta de error será una respuesta de error con el formato documentado en el apartado correspondiente del presente manual.

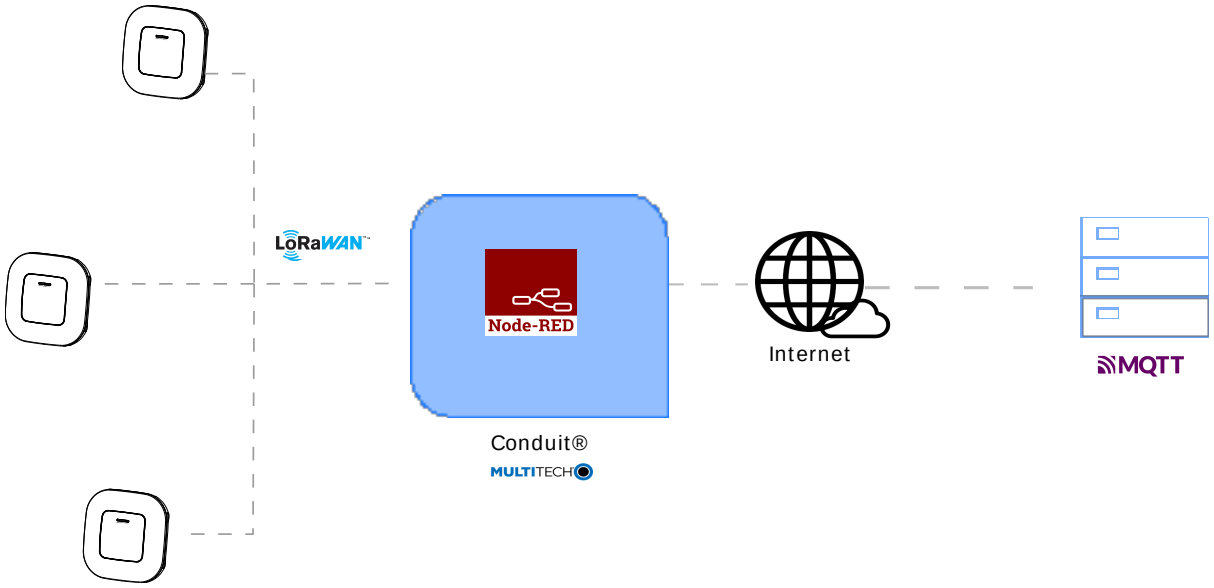
Capítulo 7. Ejemplos de integración de Nanoenvi IAQ™ LoRaWAN

7.1. Ejemplo de reenvío de datos a broker MQTT

En este ejemplo, se presenta un caso de uso de Nanoenvi IAQ™ en una red LoRaWAN desplegada con un Conduit® de la marca Multitech ejecutando el firmware mPower 5.3.8.

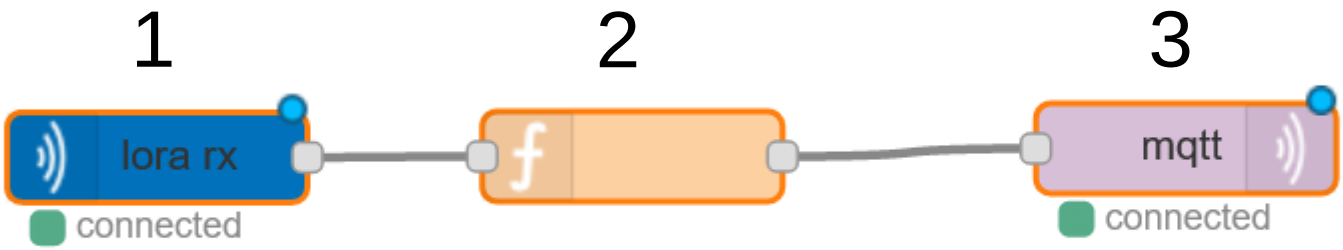
El Conduit® integra las funciones de *gateway,* Network Server y un Node-RED como *application server.* El flujo de Node-RED se configura para redirigir los mensajes con medidas de Nanoenvi IAQ™ a un broker MQTT. A continuación se muestra una imagen de la arquitectura del ejemplo:

Figura 7.1. diagrama general de red LoraWAN con redirección a Node-RED



El flujo de Node-RED se configura como indica la imagen, para que los mensajes *uplink* de los dispositivos conectados a la red LoRaWAN sean redireccionados al topic del broker MQTT `lora/uplink/<devEUI>`:

Figura 7.2. flujo NodeRED redirección de mensajes



El flujo consta de tres elementos:

1. Nodo **lora rx** que recibe los mensajes de la red LoRaWAN

2. Nodo de función. Ejecuta el siguiente código:

```
var incoming_msg = msg;
var outgoing_msg = {};
outgoing_msg.payload = msg.payload;
outgoing_msg.topic = "lora/uplink/" + msg.eui;
return outgoing_msg;
```

3. Nodo de publicación MQTT. Sólo se le configura la IP del broker MQTT.

Una vez recibidos los mensajes en el broker MQTT, se pueden suscribir servicios al topic correspondiente para recoger, deserializar y explotar los datos.

7.2. Ejemplo de deserialización de mensajes de sensores utilizando Node-RED

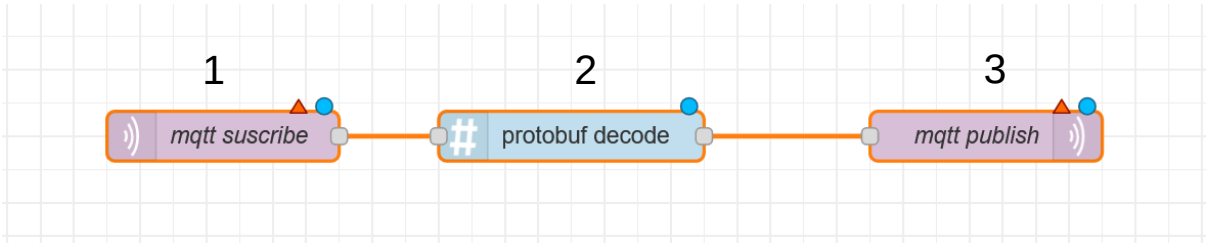
En este ejemplo se muestra cómo configurar Node-RED para que reciba mensajes suscribiéndose a un topic de un broker MQTT con medidas de Nanoenvi IAQ™ serializadas con protobuf , las convierta a un mensaje en formato JSON y las publique en otro topic del broker MQTT.

En primer lugar se requiere el nodo `nodered-contrib-protobuf` (`https://flows.nodered.org/node/node-red-contrib-protobuf`). Se puede instalar ejecutando desde línea de comandos:

```
npm install node-red-contrib-protobuf
```

Con esto y la descripción del formato de mensaje en lenguaje proto3, se puede crear el flujo mostrado en la imagen:

Figura 7.3. flujo de deserialización de mensaje protobuf

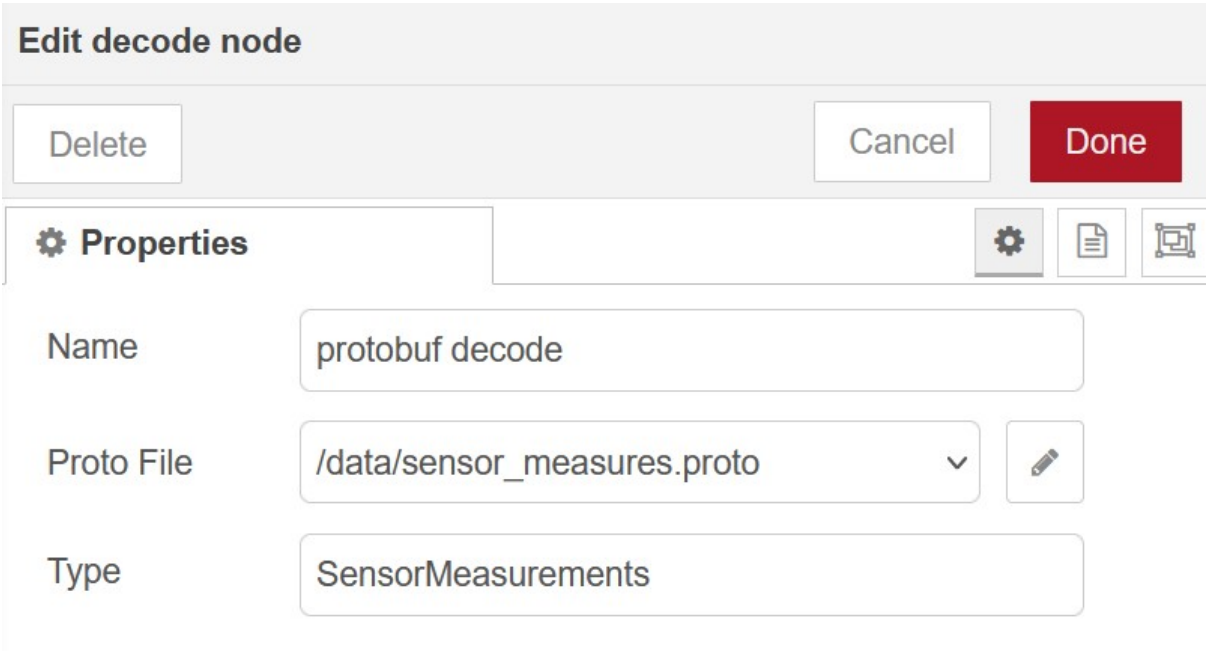


El flujo tiene los siguientes elementos:

1. Nodo de suscripción MQTT: recibe mensajes MQTT de Nanoenvi IAQ™
2. Nodo de decodificación de mensajes con medidas de Nanoenvi IAQ™.
3. Nodo de publicación de mensajes en formato JSON en broker MQTT.

El nodo protobuf decode tiene la siguiente configuración:

Figura 7.4. configuración nodo protobuf decode



El campo Proto File contiene la ruta del archivo con la descripción del formato de mensaje de medidas de Nanoenvi IAQ™ incluido en este manual.

7.3. Ejemplo de deserialización de mensajes de sensores utilizando Python

En este ejemplo se muestra cómo escribir un script en lenguaje Python que convierta el payload de un mensaje con medidas de sensores en formato protobuf en formato JSON

En primer lugar se debe instalar el paquete `grpcio-tools` ejecutando desde línea de comandos:

```
pip install grpcio-tools
```

A partir de la descripción del formato del mensaje de medidas, se debe generar la librería de Python que implementa la serialización/deserialización ejecutando el siguiente comando en el mismo directorio donde se encuentra el archivo de descripción de formato de medidas de sensores:

```
protoc --python_out=. sensor_measures.proto
```

La ejecución del comando generará un archivo llamado *sensor_measures_pb2.py* con el siguiente contenido:

```
# -*- coding: utf-8 -*-
# Generated by the protocol buffer compiler.  DO NOT EDIT!
# source: sensor_measures.proto
"""Generated protocol buffer code."""
from google.protobuf import descriptor as _descriptor
from google.protobuf import descriptor_pool as _descriptor_pool
from google.protobuf import message as _message
from google.protobuf import reflection as _reflection
from google.protobuf import symbol_database as _symbol_database
# @@protoc_insertion_point(imports)

_sym_db = _symbol_database.Default()




DESCRIPTOR = _descriptor_pool.Default().AddSerializedFile(b'\n\x15sensor_measures.proto
\"\xe3\x01\n\x12SensorMeasurements\x12\x0f\n\x07msg_ver\x18\x01 \x01(\r\x12\x0b\n
\x03\x63o2\x18\x02 \x01(\x02\x12\x0b\n\x03voc\x18\x03 \x01(\x02\x12\n\n\x02\x63o\x18\x04
 \x01(\x02\x12\r\n\x05noise\x18\x05 \x01(\x02\x12\x0c\n\x04pm10\x18\x06 \x01(\x02\x12\r
\n\x05pm2_5\x18\x07 \x01(\x02\x12\x0c\n\x04temp\x18\x08 \x01(\x02\x12\x0b\n\x03hum\x18\t
 \x01(\x02\x12\x0b\n\x03prb\x18\n \x01(\x02\x12\x0b\n\x03pm1\x18\x0b \x01(\x02\x12\x0b
\n\x03pm4\x18\x0c \x01(\x02\x12\x0c\n\x04iaqi\x18\r \x01(\x02\x12\x0b\n\x03tci\x18\x0e
 \x01(\x02\x12\r\n\x05\x65iaqi\x18\x0f \x01(\x02\x62\x06proto3')



_SENSORMEASUREMENTS = DESCRIPTOR.message_types_by_name['SensorMeasurements']
SensorMeasurements = _reflection.GeneratedProtocolMessageType('SensorMeasurements',
 (_message.Message,), {
  'DESCRIPTOR' : _SENSORMEASUREMENTS,
  '__module__' : 'sensor_measures_pb2'
  # @@protoc_insertion_point(class_scope:SensorMeasurements)
  })
_sym_db.RegisterMessage(SensorMeasurements)

if _descriptor._USE_C_DESCRIPTORS == False:

  DESCRIPTOR._options = None
  _SENSORMEASUREMENTS._serialized_start=26
  _SENSORMEASUREMENTS._serialized_end=253
# @@protoc_insertion_point(module_scope)
```

Se puede escribir un script que utilice el código generado para leer un archivo con el payload de un mensaje enviado por el dispositivo y lo convierta a JSON:

```
import argparse
import sys
import sensor_measures_pb2
from google.protobuf.json_format import MessageToDict

class NanoenviIaqMeasDeserializer:
    """ Class with method to convert Nanoenvi IAQ protobuf measures message to JSON """

    def deserialize(self, file_path):
        """ Receives as parameter the path of a file with Nanoenvi IAQ protobuf
```

```
        measures message, reads it converts it to JSON and returns it. """

        file_obj = open(file_path, 'rb')
        file_content = file_obj.read()
        file_obj.close()

        meas_obj = sensor_measures_pb2.SensorMeasurements()
        meas_string = meas_obj.ParseFromString(file_content)
        meas_dict = MessageToDict(meas_string)

        return meas_dict


if __name__ == "__main__":

    # Parse arguments
    try:
        parser = argparse.ArgumentParser()
        parser.add_argument(
            "-f",
            "--file",
            help="file containing Nanoenvi IAQ measurements serialized with protobuf",
            nargs="?",
            required=True,
        )

        args = parser.parse_args()
    except Exception as exception_object:
        print(exception_object)
        sys.exit(2)

    file_path = args.file

    # Instance NanoenviIaqMeasDeserializer
    deserializer = NanoenviIaqMeasDeserializer()
    # Deserialize file content
    result = deserializer.deserialize(file_path)
    # Print result
    print(result)
```

El script escrito se puede invocar desde línea de comandos pasando como argumento un archivo con el payload de un mensaje enviado por Nanoenvi IAQ™:

```
python main.py -f file_with_mesures_payload.bin
```

Nanoenvi