Operating instruction manual

netFIELD App Platform Connector

Hilscher Gesellschaft für Systemautomation mbH
www.hilscher.com

DOC200201OI02EN | Revision 2 | English | 2021-03 | Released | Public

Table of contents

1 Introduction

1.1 About this document

1.1.1 Description of the contents

This document describes the **netFIELD App Platform Connector** from Hilscher.

1.1.2 List of revisions

Index	Date	Author	Revision
1	2020-09-17	MKE	Document created.
2	2021-03-24	MKE	Document updated to netFIELD App version 1.4 and to netFIELD Portal version 2.2. Section *Docker Images Cleaner tab* [page 36] added.

Table 1: List of revisions

1.1.3 Conventions in this document

Notes, operation instructions and results of operation steps are marked as follows:

Notes

Important:
<important note>

Note:
<simple note>

<note, where to find further information>

Operation instructions

1. <operational step>
   - ➢ <instruction>
   - ➢ <instruction>
2. <operational step>
   - ➢ <instruction>
   - ➢ <instruction>

Results

<intermediate result>

<final result>

1.2 Terms and abbreviations

Term	Description
IIoT	Industrial Internet of Things
IT network	Information technology network
OT network	Operational technology network
netFIELD App	netFIELD application container from Hilscher, deployable via netFIELD Platform and running in the IoT Edge Docker of the netFIELD OS
netFIELD OS	Cross-platform capable operating system with connection to the netFIELD Platform
netFIELD OS Datacenter	netFIELD OS for virtual machines respectively virtualization environments
netFIELD Edge	Devices or systems running the netFIELD OS
netFIELD Platform	Internet-hosted platform providing APIs for cloud-to-cloud and cloud-to-edge communication. Basis for the netFIELD Portal
netFIELD Portal	Web-based user interface for the netFIELD Platform services
netFIELD Cloud	netFIELD Platform and netFIELD Portal
netX	Multi-protocol communication controller for OT networks

Table 2: Terms and abbreviations

1.3 Brief description

1.3.1 Overview

netFIELD App Platform Connector is a container for transporting data from your netFIELD Edge Device (or netFIELD OS Datacenter) to the netFIELD Cloud.
It also allows you to perform certain device management functions (like restarting and updating the netFIELD Operating System and deleting obsolete Docker containers) remotely from the cloud without having to login to the Local Device Manager.
The Platform Connector container runs in the **IoT Edge Docker** of the Edge Device. It subscribes to MQTT topics from a local MQTT Broker (which can also be running in the IoT Edge Docker) and forwards the messages (published by other local containers or MQTT clients) via AMQP to the netFIELD Platform.
At the platform, the message data is exposed not only as a WebSocket Secure (WSS) endpoint for real-time data streaming, but also as a HTTPS endpoint for historical data querying.

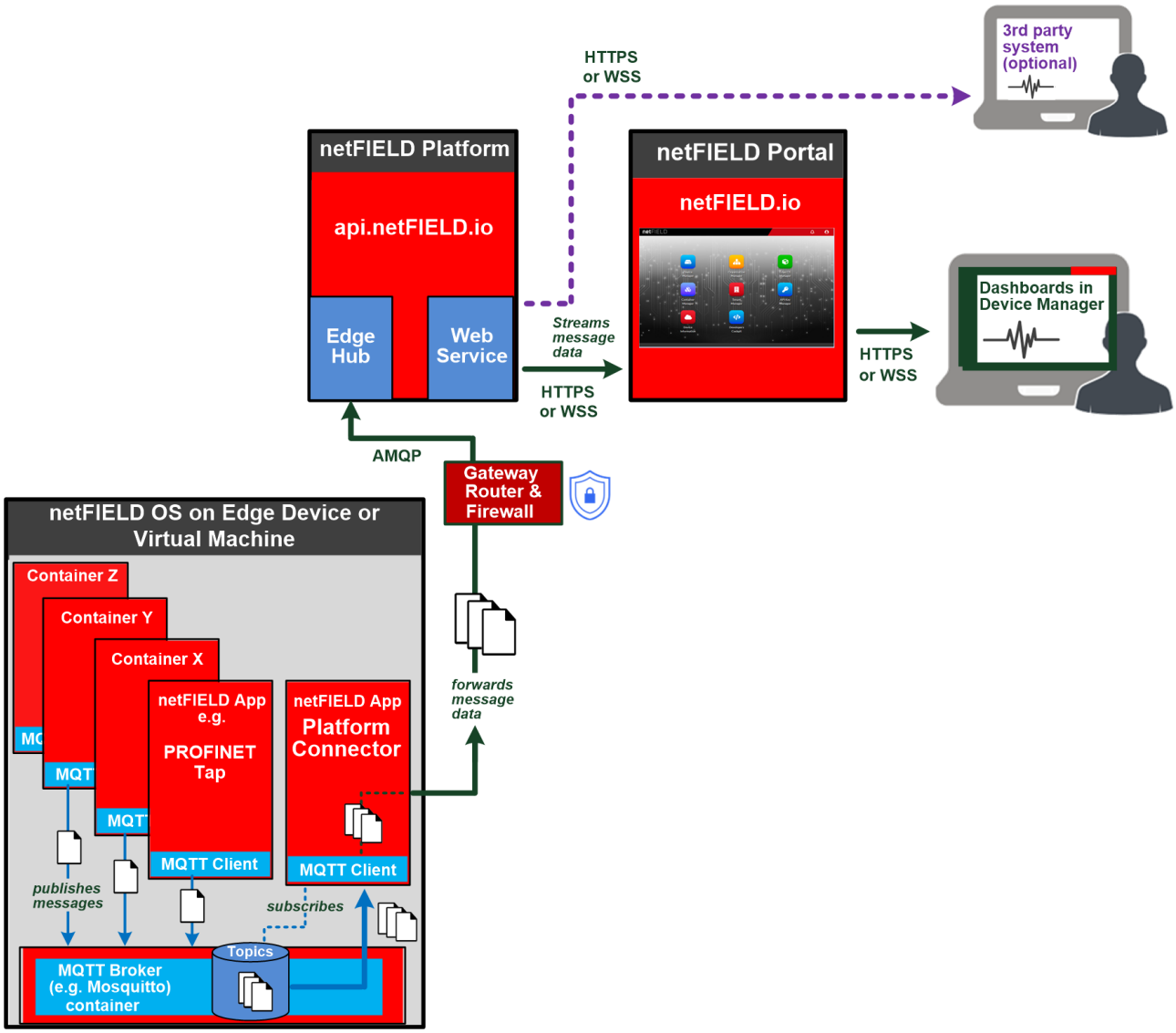

Figure 1: Platform Connector MQTT message data flow diagram

A simple code example for connecting to the netFIELD WebSocket server and consuming the message data is provided in section *Consuming message data via WebSocket (Javascript example)* [page 26].
Note that some netFIELD application containers, like e.g. the *netFIELD App Edge Monitor*, provide their own ready-to-use “plugin” dashboards in the netFIELD Portal for visualizing their subscribed message data (transmitted by the Platform Connector).

1.3.2 Local and cloud dashboards

The netFIELD App Platform Connector provides two dashboards (which are automatically “plugged-in” after container deployment):

- A dashboard in the Local Device Manager of the netFIELD OS for:
  - adding (and deleting) MQTT topic subscriptions
  - configuring MQTT settings
- A “cloud” dashboard in the Device Manager of the netFIELD Portal for:
  - performing remote netFIELD OS firmware updates
  - adding (and deleting) MQTT topic subscriptions
  - displaying message data of subscribed of MQTT topics
  - sending “Cloud to Device” MQTT messages
  - remote “cleaning” of obsolete images in IoT Edge Docker
  - performing remote netFIELD OS restart

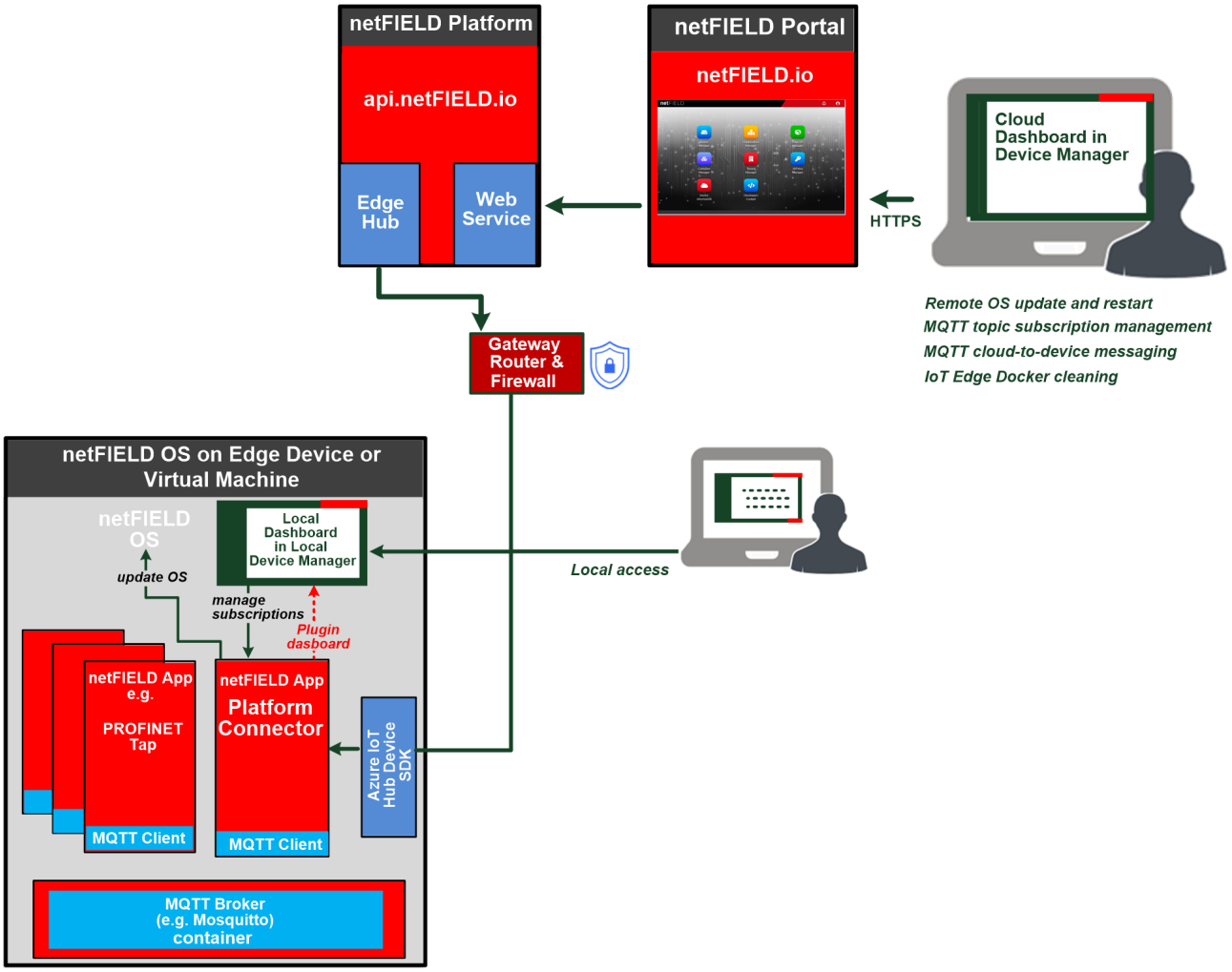

Figure 2: Platform Connector configuration

1.3.3 Requirements for using the netFIELD App Platform Connector

- netFIELD Portal account with permissions `installContainers` and `viewContainers`.
- netFIELD Edge Device or netFIELD OS Datacenter (V2.0 or higher) that is “onboarded” (i.e. “registered”) in the netFIELD Portal.
- MQTT Broker (e.g. *mosquitto*) that can be accessed from your Edge Device or Virtual Machine.
  (The MQTT Broker does not need to be deployed on the same netFIELD OS.)

2 Deployment and configuration

2.1 Deploying Platform Connector and MQTT Broker container

This section describes how to deploy (“install”) the netFIELD App Platform Connector and MQTT Broker containers on your Edge Device respectively netFIELD OS.

Note:
This manual deployment is only necessary if the containers have not already been deployed via *Deployment Manifest* during device onboarding.
Check the **Installed Containers** tab (Device Manager > [your device] > DEVICE NAVIGATION > Containers > Installed Containers) to see whether the containers have already been deployed and are running on the device.

We recommend you to use the open source MQTT Broker *Mosquitto*, which is pre-configured to be used on your Edge Device/netFIELD OS with the Platform Connector. (It is preset in the Default General MQTT settings of the netFIELD OS).
The *Mosquitto* container is ready to use and available in the netFIELD Portal for deployment on your device.

➢ Select your device in the Portal’s **Device Manager** and open the **Available Containers** tab in the DEVICE NAVIGATION (DEVICE NAVIGATION > Containers > Available Containers).

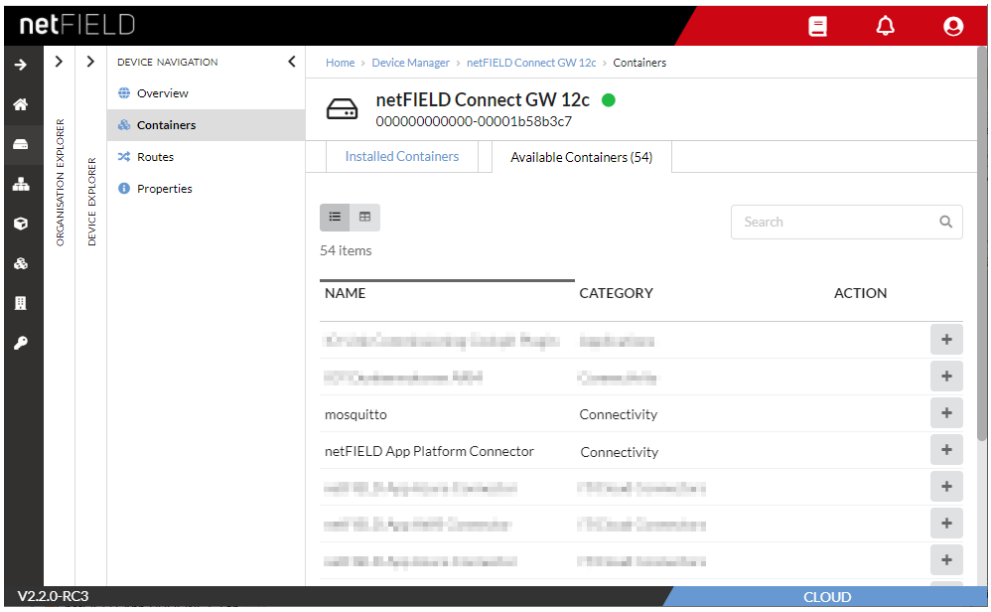

Figure 3: Available Containers tab

Deploy MQTT Broker (Mosquitto)

➢ Scroll through the list and look for the **mosquitto** container (or type its name into the **Search** field).

➢ Click on the **mosquitto** entry or on the **+** button.

↬ The container deployment dialog screen opens:

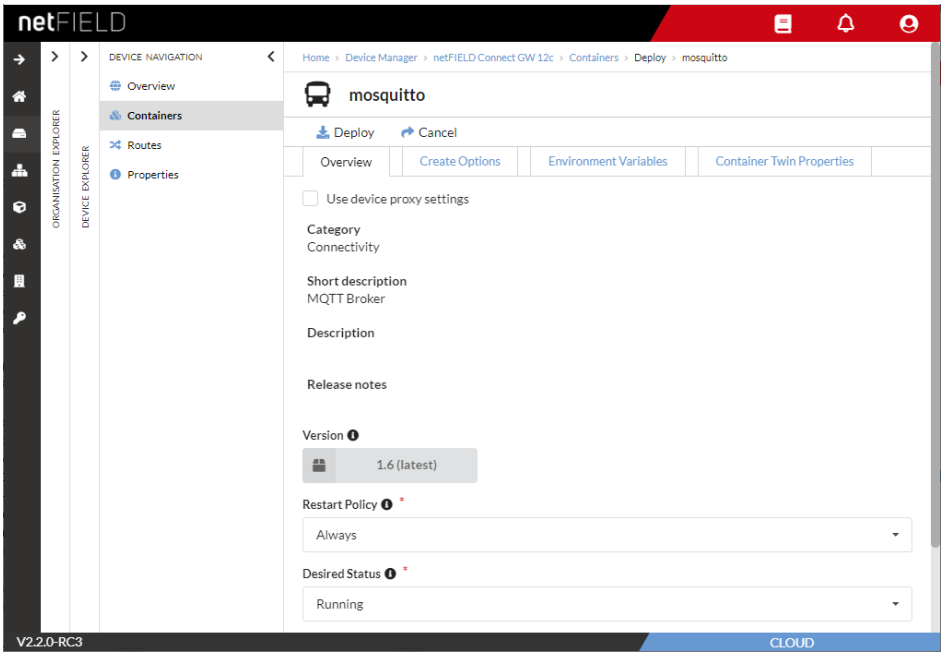

Figure 4: Deploy mosquitto

➢ In the **Overview** tab, keep the default settings.

Important:
We strongly recommend you to keep also the default settings in the **Create Options**, **Environment Variables**, and **Container Twin Properties** tabs.
If necessary, these configuration settings can be changed later (i.e. after having deployed the container). Note that only expert users should change these settings.

➢ Click **Deploy** button.

↬ The mosquitto container image is downloaded from the cloud to the device and automatically started on the device. This may take a few minutes. After its deployment, the container will from now on be listed in the **Installed Containers** tab of your device.

Deploy Platform Connector

- Scroll through the list of the **Available Containers** tab and look for the **netFIELD App Platform Connector** container (or type its name into the **Search** field).
- Click on the **netFIELD App Platform Connector** entry or on the **+** button.

The container deployment dialog screen opens:

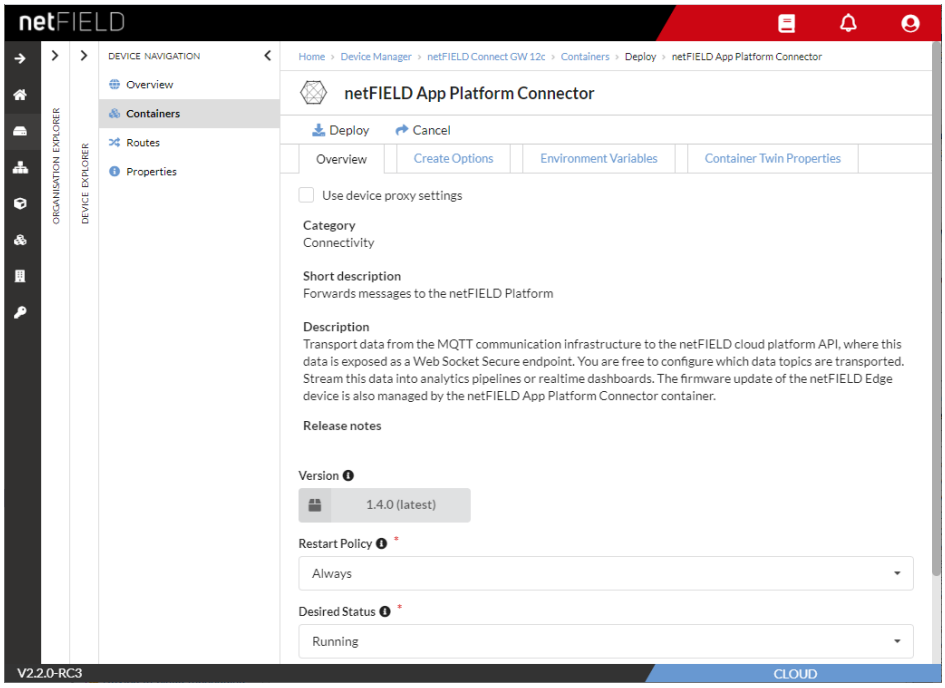

Figure 5: Deploy netFIELD App Platform Connector container

- In the **Overview** tab, select the **Use device proxy settings** option if your device is configured to use HTTP or HTTPS Proxy servers. (Whether your device is configured for Proxy Servers or not is indicated on the **Overview** page of your device.)
- Keep the **Version**, **Restart Policy** and **Desired Status** options on their pre-configured default settings; i.e. **Restart Policy** = **Always** and **Desired Status** = **Running**. (This ensures that the latest version of the container will be deployed and that it will be automatically started on the device after deployment.)

Important:
We strongly recommend you to keep also the default **Create Options**, **Environment Variables** and **Container Twin Properties** settings.
If necessary, you can change these configuration settings later (i.e. after having deployed the container). Note that changing these settings is for expert users only.

- Click **Deploy** button.

The Platform Connector container image is downloaded from the cloud to the device and automatically started on the device. This may take a few minutes.

➢ In the **Installed Containers** tab, check if the Platform Connector container is in Status “Connected” (green dot).

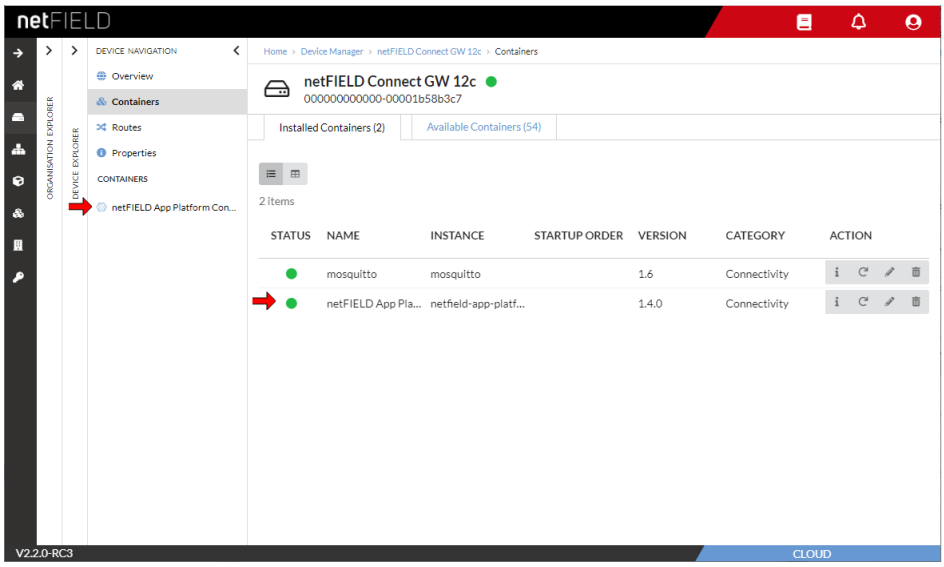

Figure 6: Deployed netFIELD App Platform Connector container

Note that the DEVICE NAVIGATION now features the **netFIELD App Platform Connector** plug-in.

➢ Click on the **netFIELD App Platform Connector** entry in the DEVICE NAVIGATION to open the Platform Connector’s dashboard, where you can e.g. subscribe to MQTT topics.

2.2 Configuration

2.2.1 Manage MQTT topics

Although MQTT topic subscriptions are stored “locally” in the Platform Connector container running on the Edge Device, you can nevertheless manage your subscriptions conveniently from the Platform Connector dashboard in the Portal.
Select your device in the Portal’s Device Manager and open the **Topics** tab of the Platform Connector dashboard (DEVICE NAVIGATION > netFIELD App Platform Connector > Topics).

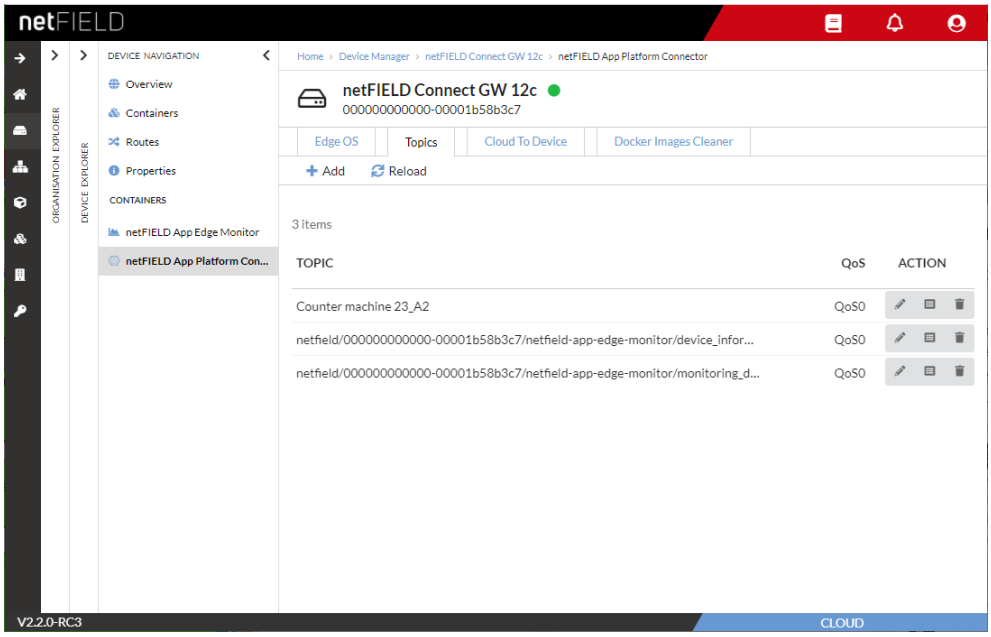

Figure 7: Topics tab

Element	Description	
Add	Opens a dialog for entering a new MQTT topic subscription (see section *Subscribing to MQTT Topics* [page 15]).	
Reload	Reloads the list of topic subscriptions from the device.	
Topic	Subscribed MQTT topic.	
QoS	MQTT Quality of Service: QoS0: at most once QoS1: at least once QoS2: exactly once	
ACTION		Update Topic: Opens a dialog for changing the Quality of Service of the subscription. **Note**: You cannot edit the topic itself, only its QoS. If you want to change a topic name string, you must delete it and add it as a new topic.
		View Data: Displays current message data of the subscribed topic. See section *View message data of subscribed MQTT Topics in Portal* [page 24].
		Delete Topic: Deletes the subscription.
« ‹ 1 2 › »	If the list contains more than ten entries, you can scroll here to display the next ten items.	

Table 3: Elements on Topics tab

2.2.2 Subscribing to MQTT Topics

You can add new MQTT topic subscriptions for the Platform Connector app either from the cloud (i.e. from the dashboard in the netFIELD Portal) or in the local dashboard (i.e. in the Local Device Manager).

Note:

If you have deployed the **netFIELD App Edge Monitor** on your Edge Device, note the following:
The **Edge Monitor** container communicates its topics via a special API function to the Platform Connector, and via this function, the topics are automatically added to the subscriptions list of the Platform Connector. Thus, if you are using the **Edge Monitor**, you do not have to enter these topics manually. After deploying the **Edge Monitor** container, you will find the following two topics in the Platform Connector dashboard:
`netfield/[gateway prefix]/netfield-app-edge-monitor/device_information`
and
`netfield/[gateway prefix]/netfield-app-edge-monitor/monitoring_data`

Requirements

- The netFIELD App Platform Connector container has been deployed and is running on your device.
- An MQTT Broker can be accessed from your device.
- You know the MQTT topic(s) to which you want to subscribe. Consult the commissioning engineer (or the documentation) of the application container to find out what topic(s) the container provides (i.e. what topic(s) it publishes to the MQTT Broker).

Subscribing via dashboard in netFIELD Portal (“cloud dashboard”)

➢ In the **DEVICE NAVIGATION**, click on **netFIELD App Platform Connector** to open its dashboard, then open the **Topics** tab.

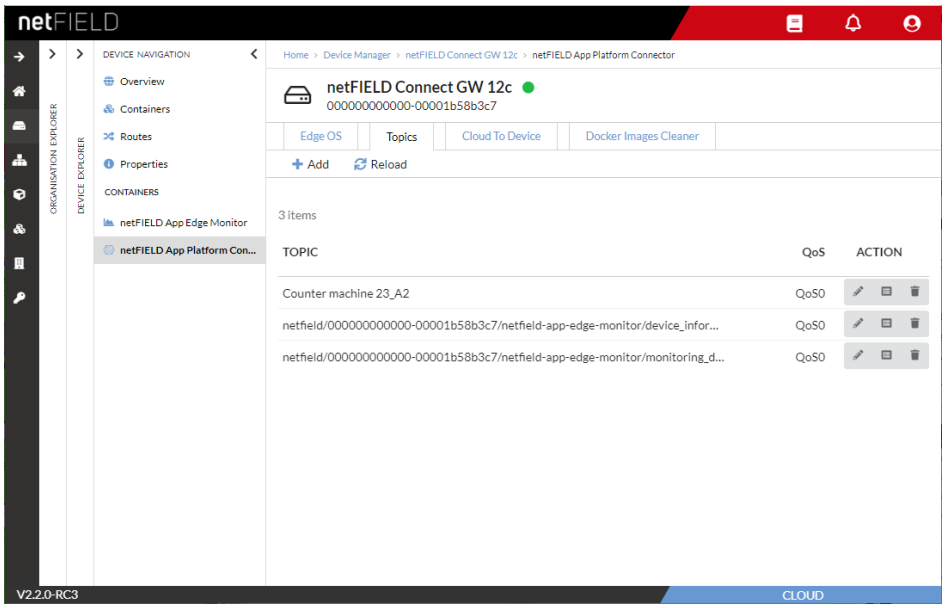

Figure 8: Create new topic

➢ Click **+ Add** button.

➢ In the **Topic** field, enter the string of the topic to which you want to subscribe.
(Use copy & paste via your clipboard whenever possible.)

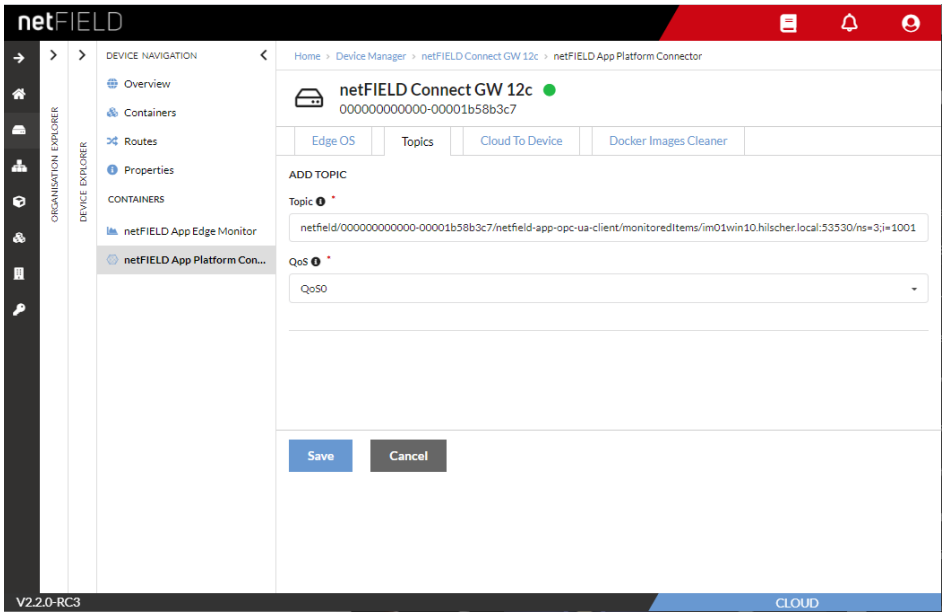

Figure 9: Create new topic

➢ In the **QoS** drop-down list, select the MQTT Quality of Service (default is QoS0):
QoS0: At most once
QoS1: At least once
QoS2: Exactly once

Note:
This is the QoS of the message delivery from the MQTT broker to the subscribing client, i.e. to the Platform Connector on the device. If you define here a QoS that is lower than the QoS defined by the publishing container, the MQTT broker transmits the message with the lower QoS.
Note that the QoS does not relate to the messages that are being sent from the Platform Connector to the netFIELD cloud.

➢ Click **Save** button to add the topic.

➪ The **Success – Topic added** notification appears. Note that this indicates only that the subscription has been saved in the Portal's dashboard and that it will be transmitted to the Platform Connector container on the device. It does not indicate whether the subscription has been actually successful (i.e. acknowledged by the broker) or not.

⇨ After saving, the topic subscription is transmitted from the Portal to the Platform Connector container in the Edge Device. The Platform Connector (i.e. the MQTT client within the Platform Connector container) then sends a SUBSCRIBE message for this topic to the MQTT Broker.

Subscribing via dashboard in Local Device Manager (local dashboard)

You can also add your subscriptions in the "local" Platform Connector dashboard (i.e. in the dashboard that is provided in the Local Device Manager of the netFIELD OS).
You can connect to the Local Device Manager from your LAN (by entering the IP address of your Edge Device/netFIELD OS in your browser) or via **Remote Control** function from the netFIELD Portal.
For more information about the Remote Control function, see section *Remote Control* in the operating instruction manual *netFIELD Portal* (DOC190701OIxxEN).

Note:
You must know the login credentials (user name and password) of the Local Device Manager.

➢ Login to the Local Device Manager.

➢ In the navigation panel of the Local Device Manager, select **netFIELD APP Platform Connector** entry.

➮ The **Platform Connector** dashboard opens:

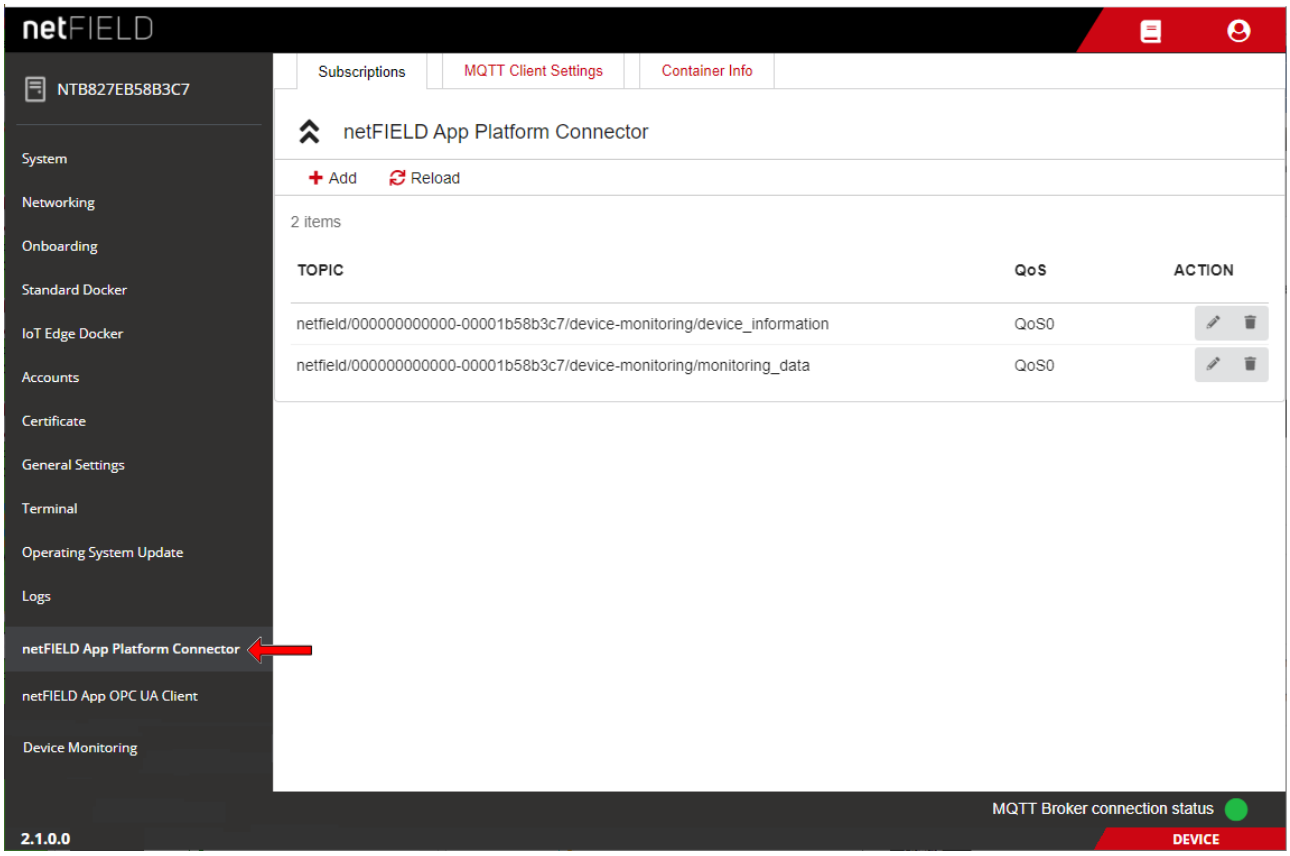

Figure 10: Platform Connector in Local Device Manager

➢ In the **Subscriptions** tab, click **+ Add** button.

➢ In the **Topic** field, enter the string of the topic to which you want to subscribe.

➢ In the **QoS** dropdown-list, select the MQTT Quality of Service.

Note:
This is the QoS of the message delivery from the MQTT broker to the subscribing client, i.e. to the Platform Connector. If you define here a QoS that is lower than the QoS defined by the publishing container, the MQTT broker transmits the message with the lower QoS.
Note that the QoS does not relate to the messages that are being sent from the Platform Connector to the netFIELD cloud.

➢ Click **Save** button to add the topic.

⇨ The topic is added to the list and the Platform Connector (i.e. the MQTT client within the Platform Connector container) sends a SUBSCRIBE message for this topic to the MQTT Broker. The new topic subscription will also be automatically added to the subscriptions list in the cloud dashboard.

2.2.3 Updating and deleting MQTT Topic Subscriptions

Update topic

You can change the Quality of Service of a subscription by clicking the  button next to the topic in the **Topics** tab.
Note however, that you cannot edit the topic itself, only its QoS.
If you want to change a topic name string, you must delete it and add it as a new topic.

Delete topic

You can delete a subscription by clicking the  button next to the topic in the **Topics** tab.

2.2.4 Changing MQTT Client default settings (optional)

In the **MQTT Client Settings** tab of the **Platform Connector** dashboard in the Local Device Manager, you can customize the MQTT client settings of the Platform Connector. By default, the Platform Connector container uses the standard MQTT client settings of the netFIELD OS, which can be viewed (and changed) in the Local Device Manager under **General Settings** > **Default MQTT Client Settings**.

If you want to use different settings for your Platform Connector – e.g. if you want to use a different broker than the preset `tcp://mosquitto:1883` – you can uncheck the **Use General Settings** option and enter your new parameters in the configuration fields that are now displayed:

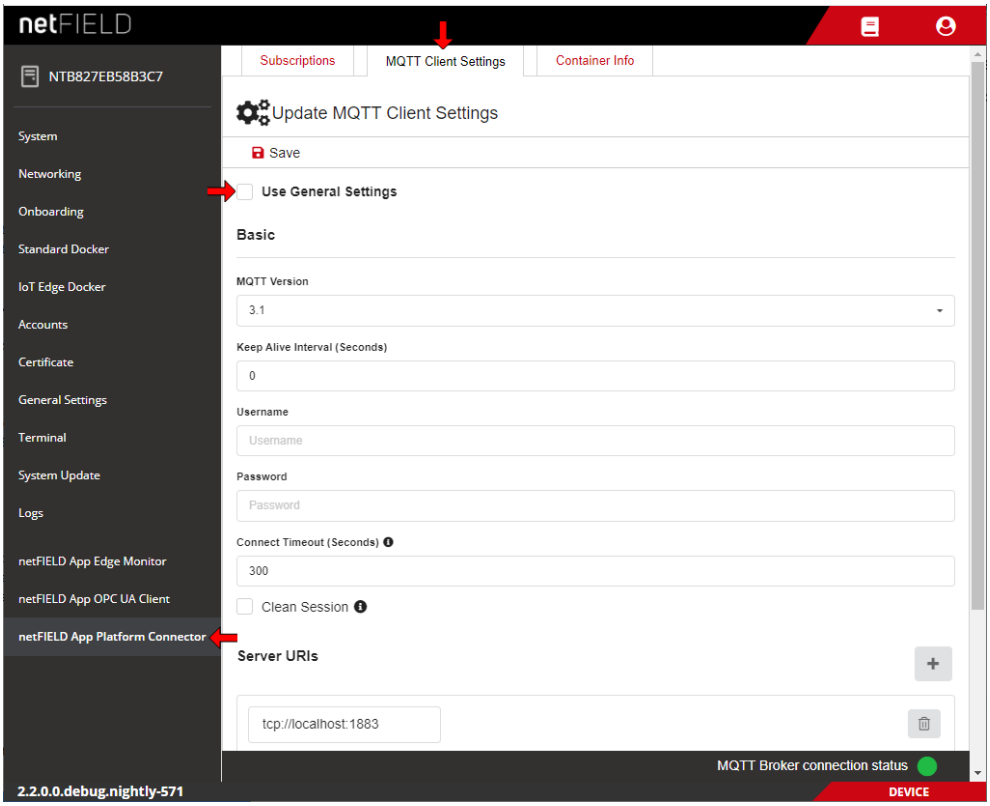

Figure 11: MQTT Client settings of Platform Connector in local Device Manager

Note:
Note that you must restart the Platform Connector container in order to apply the new settings (see subsection below).

Note:
Changes to the MQTT Client Settings that you make here for your Platform Connector app will not affect the standard “global” MQTT Settings of your netFIELD OS in the Local Device Manager under **General Settings** > **Default MQTT Client Settings**.

Element	Description	
MQTT Version	MQTT version to be used (depending on the MQTT Broker).	
Keep Alive Interval	Defines the maximum length of time in seconds that the broker and client may not communicate with each other.	
Username	User name for authentication at the Broker (if implemented and required by the Broker). Note that the Mosquitto Broker from the netFIELD Portal does not require login authentication.	
Password	Password for authentication at the Broker (if implemented and required by the Broker). Note that the Mosquitto Broker from the netFIELD Portal does not require login authentication.	
Connect Timeout	Defines the maximum length of time in seconds that is allowed for completing the connection process.	
Clean Session	If **Clean Session** is selected, the client does not want a persistent session (meaning that if the client disconnects for any reason, all information and messages that are queued from a previous persistent session are lost. If **Clean session** is unchecked, the broker creates a persistent session for the client.	
Server URIs	Server URI of the MQTT Broker **Note**: When multiple server URIs are specified, the client will try to connect to each server one after the other, starting with the first server in the list. If a server connection was established successfully, only this connection will be used. The client will not open multiple connections to multiple servers simultaneously.	
Last Will and Testament	Select this option if you want to use the “last will and testament” (LWT) feature of MQTT. (I.e. to notify other clients about an unexpected loss of connection to the broker)	
	Topic name	Topic name of LWT message
	Retained	“Retained” flag of LWT message
	Quality of Service	QoS of LWT message
	Message	Message text, e.g. “unexpected loss of connection”
SSL	Select this option if you want to use SSL encryption for creating a secure connection to the MQTT Broker. **Note**: This option is for expert users only! In the standard use case, in which the Mosquitto Broker and the Platform Connector container are running on the same device, a secure SSL/TLS connection is not necessary (the overhead of the secure connection can thus be avoided).	
	File name and path to private key in PEM format	Enter here the complete path to the private key on the device.
	File name and path to certificate chains in PEM format	Enter here the complete path to the certificate chains on the device
	Override the trusted CA certificates in PEM format	Enter here the complete path to override the trusted CA certificates on the device
	Enable verification of the server certificate	If this option is disabled, the Platform Connector will also accept invalid certificates from the Broker (not recommended).

Table 4: MQTT Client Settings

➢ Click **Save** button to save your new MQTT Client Settings.

Restart container and check new settings

After changing the settings – e.g. by defining a new MQTT Server (Broker) URI – you have to restart the Platform Connector container manually:

➢ In the Local Device Manager, open the **IoT Edge Docker**.

➢ Under **Containers**, expand the **netFIELD App Platform Connector** entry and click the **Restart** button.

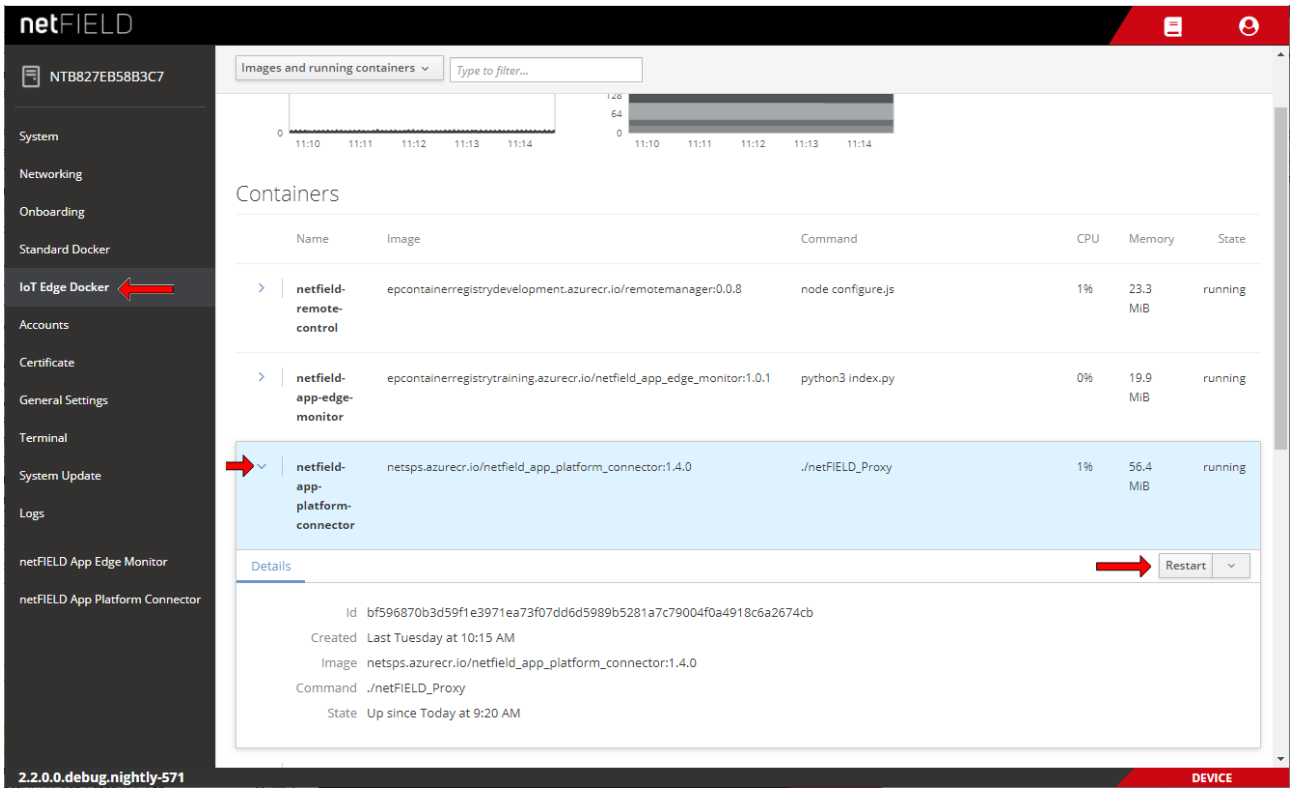

Figure 12: Restart container

➢ The Platform Connector app restarts. This can take a few seconds.

➢ Go back to the Platform Connector dashboard and check the **MQTT Broker connection status** indicator in the footer to see if the connection to the new server has been successfully established:

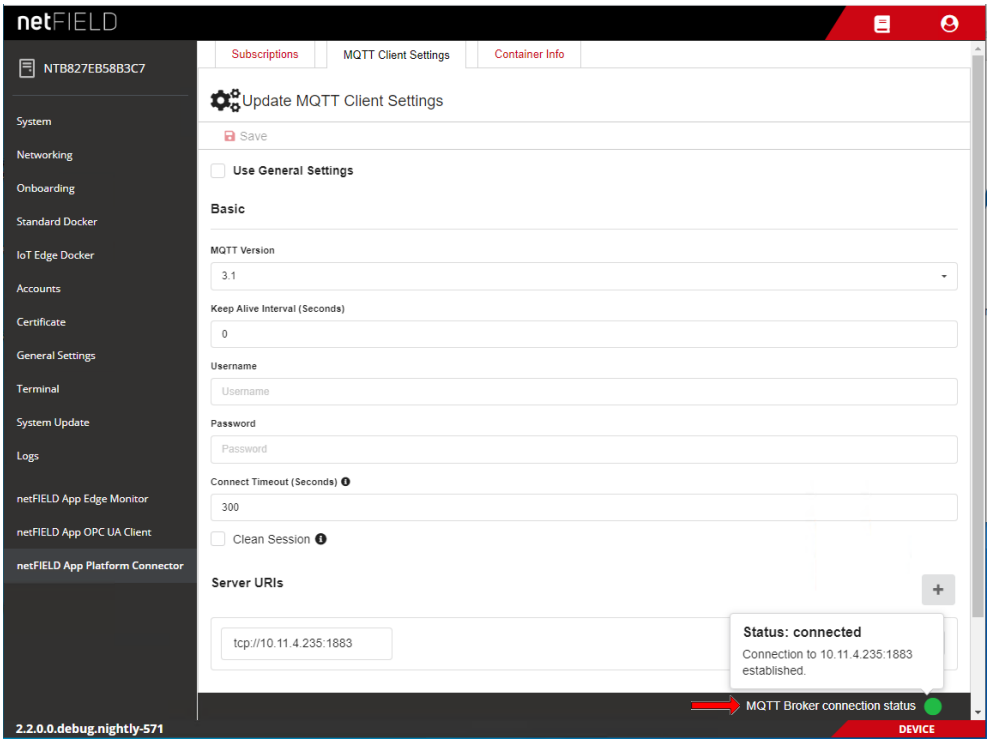

Figure 13: MQTT server connection status indicator in footer

3 Device-to-cloud messaging (telemetry)

3.1 View message data of subscribed MQTT Topics in Portal

In the Platform Connector dashboard in the netFIELD Portal, you can view the current topic messages that the Platform Connector is sending to the netFIELD cloud. The messages are presented as JSON objects with “transparent” payload data. The messages are “refreshed” at the interval determined by the publishing MQTT Client.

➢ In the **DEVICE NAVIGATION**, click on **netFIELD App Platform Connector** to open its dashboard, then open the **Topics** tab.

➢ To display the data, click on the topic in the **Topics** tab or click the (View Data) button next to the topic.

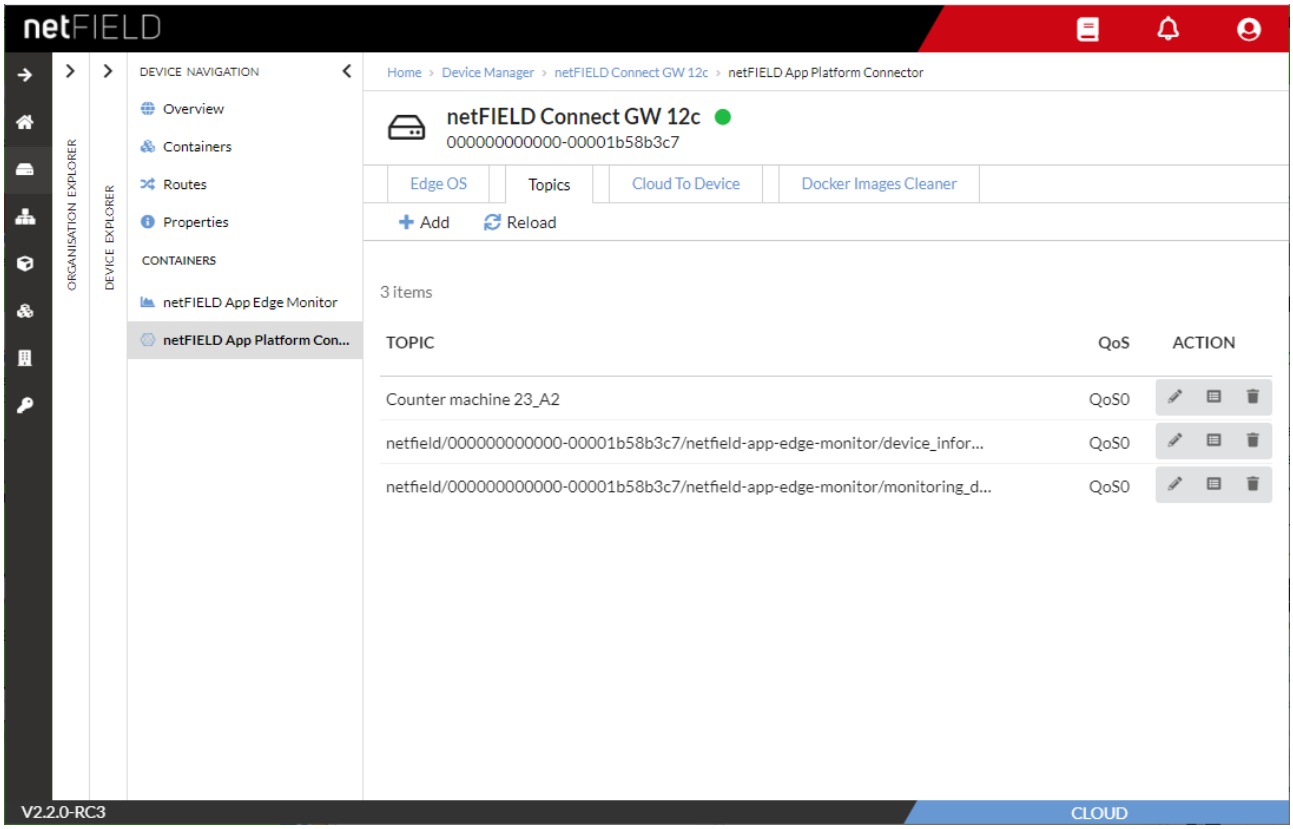

Figure 14: Topic subscriptions

The current message data stream is displayed. It is refreshed at the interval that was determined by the publishing MQTT Client (in this example, it is the data published by the **netFIELD App OPC UA Client** container):

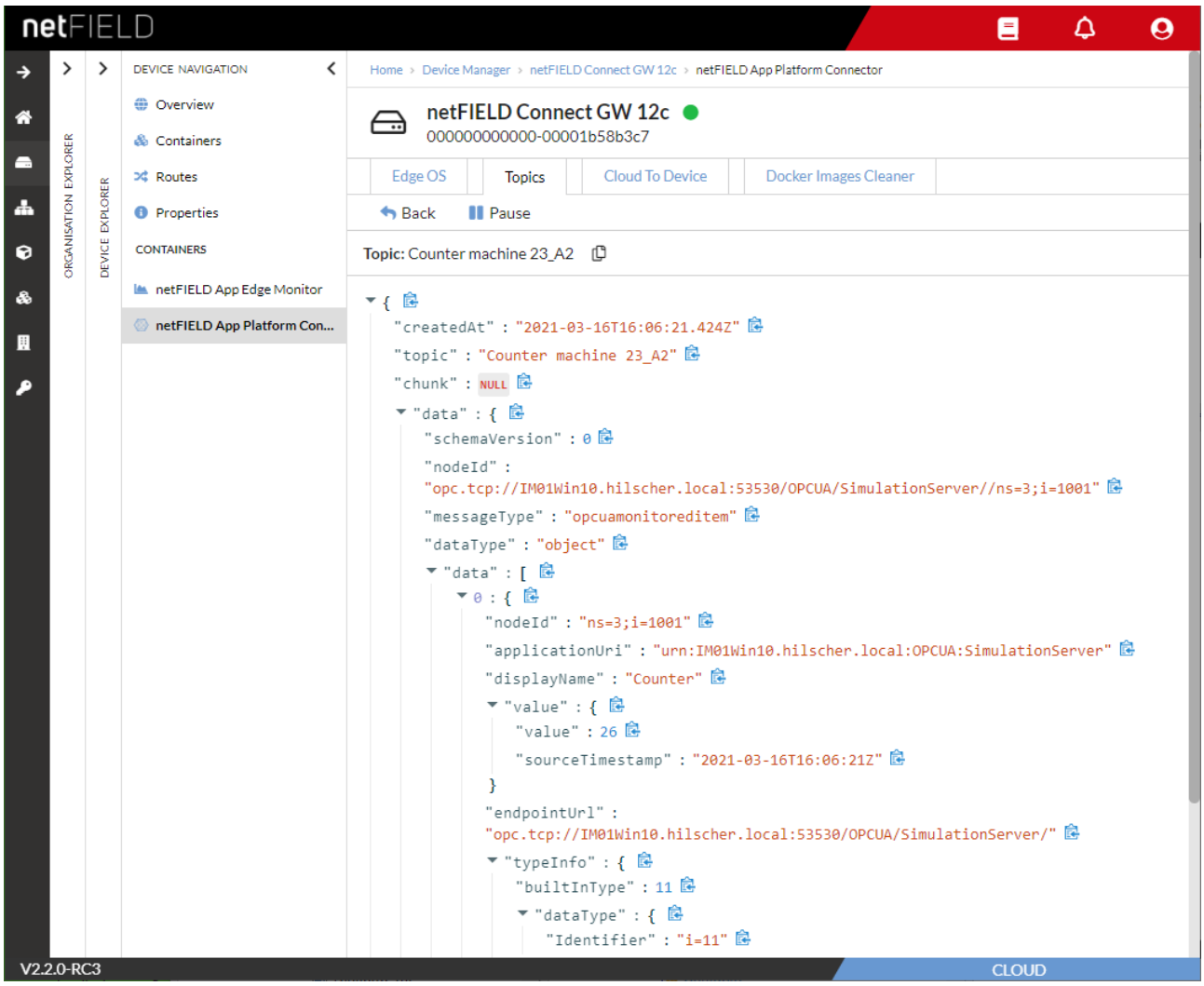

Figure 15: Example of message data

The data stream page features the following control elements for navigation and inspection of the message data:

Control element	Description
Elements in toolbar	
Pause	“Freezes” the currently shown data set
Resume	Resumes to display the “live” data stream
Back	Brings you back to the topic subscriptions list
	Copies the **Topic** name string to your clipboard
Elements in data set	
	Click on the arrow to display subordinate data elements (expand the structure)
	Click on the arrow to hide subordinate data elements (collapse the structure)
	Click on the icon to copy the data element and subordinate elements to your clipboard
	Indicates that the data element has been copied to the clipboard

Table 5: Control elements on data stream page

3.2 Consuming message data via WebSocket (Javascript example)

The published MQTT data that the Platform Connector transports from the netFIELD Edge Device to the netFIELD Platform can be accessed and consumed by a customer's application or web browser via WebSocket.

This section provides a code example of a simple program in *Javascript* for connecting to the WebSocket server of the netFIELD Platform and for consuming the data streamed from the Platform Connector.

Note that the Platform Connector automatically "chunks" messages bigger than 1MB (messages smaller than 1MB are sent "as is"). This code example does not feature any de-chunking mechanism and is therefore suited only for consuming messages that are smaller than 1MB (i.e. messages that are not "chunked").

```
/* Polyfill Websocket and btoa */
if (typeof WebSocket !== 'function') {
  // WebSocket is not globally defined, probably running in node.js
  var WebSocket = require('ws');
}
if (typeof btoa !== 'function') {
  // btoa is not globally defined, probably running in node.js
  var btoa = require('btoa');
}


/**
* WebSocket client to communicate with the netFIELD App Platform
* Connector WebSocket.
*
* @class NFAPlatformConnectorWebSocketClient
*/
class NFAPlatformConnectorWebSocketClient {
  /**
   *Creates an instance of NFAPlatformConnectorWebSocketClient.
   * @param {string} endpoint - WebSocket endpoint,
   * e.g. wss://api.netfield.io/v1
   * @param {string} authorization - Access token or API key.
   * @param {string} deviceId - deviceId of the device running
   * netFIELD App Plattform Connector.
   * @param {string} topic -
   *   topic to subscribe to (plaintext, converted to base64
   *   automatically)
   * @param {Object} [handlers] - handlers for events and messages
   * @param {function} [handlers.pubMessageHandler=console.log] -
   *   Handler to be called on receiving a netFIELD App Platform
   *   Connector update message on the WebSocket.
   * @param {function} [handlers.errorHandler=console.error] -
   *   Handler to be called on errors.
   * @param {function} [handlers.closeHandler=console.log] -
   *   Handler to be called on closing the connection.
   * @param {function} [handlers.unexpectedMessageHandler
   * =console.warn] -
   *  Handler to be called on receiving an unexpected message.
   */
  constructor(
    endpoint,
    authorization,
    deviceId,
    topic,
    {
      pubMessageHandler = console.log,
```

```
    errorHandler = console.error,
    closeHandler = console.log,
    unexpectedMessageHandler = console.warn,
  },
) {
  this.endpoint = endpoint;
  // generate a clientId
  this.clientId = (Math.random() + 1).toString(36).substring(7);
  this.deviceId = deviceId;
  this.topic = topic;
  this.authorization = authorization;
  this.pubMessageHandler = pubMessageHandler;
  this.errorHandler = errorHandler;
  this.closeHandler = closeHandler;
  this.unexpectedMessageHandler = unexpectedMessageHandler;
  this.wsClient = this._initializeWebSocketClient();

  this.subscribeToTopic = this.subscribeToTopic.bind(this);
  this.send = this.send.bind(this);
  this.sendObject = this.sendObject.bind(this);
  this.close = this.close.bind(this);
}

/**
 * Initialize the WebSocket client.
 *
 * @access private
 *
 * @returns { WebSocket } WebSocket client.
 */
_initializeWebSocketClient() {
  const client = new WebSocket(this.endpoint);
  client.onmessage = this._messageHandler.bind(this);
  client.onerror = this.errorHandler;
  client.onclose = this.closeHandler;
  client.onopen = this._sayHello.bind(this);
  return client;
}

/**
 * Handler to be invoked on receiving a message on the WebSocket.
 *
 * @param { WebSocket.MessageEvent } event -
 * WebSocket message event.
 * @param { WebSocket.Data } event.data - WebSocket message data.
 *
* @access private
 */
_messageHandler({ data }) {
  try {
    const dataObj = JSON.parse(data);
    const { type, message, payload } = dataObj;
    if (payload && payload.error) {
      this.errorHandler(data);
      return;
    }
    switch (type) {
      case 'hello':
        // got a 'hello' response after successfully
        // authenticating, subscribing
        this.subscribeToTopic(this.deviceId, this.topic);
        break;
      case 'sub':
        // got a 'sub' response after successfully subscribing
        // -> do nothing and wait for 'pub' messages
        break;
      case 'ping':
        // got a keep-alive 'ping' heartbeat from the server
        this._respondToHeartbeatPing();
```

```
          break;
        case 'pub':
          // got a 'pub' message from the server
          this.pubMessageHandler(message);
          break;
        default:
          this.unexpectedMessageHandler(data);
          break;
      }
    } catch (error) {
      this.errorHandler(error);
    }
  }

  /**
   * Subscribe to netFIELD App Platform Connector messages
   * for the given device on the given topic.
   *
   * @param {string} deviceId - deviceId of the device running
   * netFIELD App Platform Connector.
   * @param {string} topic - topic to subscribe to (plaintext,
   * converted to base64 automatically)
   */
  subscribeToTopic(deviceId, topic) {
    const topicAsBase64 = btoa(topic);
    const subscribePayload = {
     id: this.clientId,
     path: `/devices/${deviceId}/platformconnector/${topicAsBase64}
`,
     type: 'sub',
    };
    this.sendObject(subscribePayload);
  }

  /**
   * Send a string message.
   *
   * @param {string} dataString - string to send.
   */
  send(dataString) {
    const { wsClient } = this;
    if (wsClient && wsClient.readyState === wsClient.OPEN) {
      wsClient.send(dataString);
    }
  }

  /**
   * Send a message by passing in an object which will be
   * serialized before sending.
   *
   * @param {Object} dataObj - data object to send.
   */
  sendObject(dataObj) {
    this.send(JSON.stringify(dataObj));
  }

  /**
   * Close the connection to the WebSocket.
   *
   * @param {number} [code]
   * @param {string} [data]
   */
  close(code, data) {
    const { wsClient } = this;
    if (wsClient && wsClient.readyState === wsClient.OPEN) {
      wsClient.close(code, data);
    }
  }
```

```
  /**
   * Send a 'hello' message according the nes protocol
   * which authenticates this client.
   *
   * https://github.com/hapijs/nes/blob/master/PROTOCOL.md#Hello
   *
   * @access private
   */
  _sayHello() {
    const helloPayload = {
      type: 'hello',
      auth: {
        headers: {
          authorization: this.authorization,
        },
      },
      id: this.clientId,
      version: '2',
    };
    this.sendObject(helloPayload);
  }

  /**
   * Send a heartbeat keep-alive ping response according to
   * the nes protocol.
   *
   * https://github.com/hapijs/nes/blob/master/PROTOCOL.md#Heartbeat
   *
   * @access private
   */
  _respondToHeartbeatPing() {
    const pingResponsePayload = {
      id: this.clientId,
      type: 'ping',
    };
    this.sendObject(pingResponsePayload);
  }
}

// usage example

/**
* Handler to be called on receiving a netFIELD App Platform
* Connector update message on the WebSocket.
*
* @param {Object} platformConnectorMessage - netFIELD App
* Platform Connector update message object
* @param {number} platformConnectorMessage.createdAt - unix
* timestamp in milliseconds
* @param {string} platformConnectorMessage.topic - topic
* (plain text, not base64-encoded)
* @param {string} platformConnectorMessage.data - the message
* content
*/
const myPubMessageHandler = (platformConnectorMessage) => {
  // do something with the received message
  console.log('Received a netFIELD App Platform Connector:',
platformConnectorMessage);
};

/**
* Handler called on errors.
*
* Error causes:
* * WebSocket connection errors.
* * Error parsing a received message.
* * Invalid credentials.
*
* @param {*} error
```

```
*/
const myErrorHandler = (error) => {
  // do something on receiving an error
  console.error('An error occured:', error);
};

/**
* Handler to be called on receiving an unexpected message.
*
* @param {string} message
*/
const myUnexpectedMessageHandler = (message) => {
  // do something on receiving an unexpected message
  console.warn('Received an unexpected message:', message);
};

/**
* Handler to be called on closing the connection.
*
* @param {WebSocket.CloseEvent} event
*/
const myCloseHandler = (event) => {
  console.log('Connection to WebSocket closed:', event);
};

const handlers = {
  pubMessageHandler: myPubMessageHandler,
  errorHandler: myErrorHandler,
  unexpectedMessageHandler: myUnexpectedMessageHandler,
  closeHandler: myCloseHandler,
};

const endpoint = 'wss://api.netfield.io';
const deviceId = '{deviceId}';
const topic = '{topic}';
const authorization = '{authorization}';
// API Key or User Token with viewDeviceDetails permission

console.log(
  `Initializing connection to WebSocket at ${endpoint} and
subscribing to topic ${topic} on device ${deviceId} ...`,
);

const client = new NFAPlatformConnectorWebSocketClient(
  endpoint,
  authorization,
  deviceId,
  topic,
  handlers,
);

const keepConnectionOpenForSeconds = 60;
console.log(`Connection initialized, keeping open for
${keepConnectionOpenForSeconds} s ...`);
setTimeout(client.close, keepConnectionOpenForSeconds * 1000);
```

4 Cloud-to-device messaging (control)

4.1 netFIELD OS Update

The netFIELD App Platform Connector allows you to update the operating system (netFIELD OS) on your Edge Device from the cloud.

Note that this update function of the Platform Connector does not require a running MQTT Broker on the device.

Requirements

- Your Edge Device (respectively netFIELD OS Datacenter) is connected to cloud, i.e. the device is marked with a green status dot in the Portal's **Device Manager**.
- You have deployed the netFIELD App Platform Connector container on your device and the container is "connected" (the container is marked with a green status dot in the **Installed Containers** tab).
- You have the permissions `updateEdgeos` and `viewDeviceDetails`.
- A new netFIELD OS version is available.

Note:
"Downgrading" the netFIELD OS on your device via the Platform Connector is not possible. The installation of an OS version that is "lower" than the currently installed OS version will be denied.

Step-by-step instructions

➢ Select your device in the Portal's Device Manager and open the **Edge OS** tab of the Platform Connector dashboard (DEVICE NAVIGATION > netFIELD App Platform Connector > Edge OS).

➮ The upper area of the tab shows your currently installed **Firmware Version** (i.e. netFIELD OS version) and the **Model Name** and the **Hardware ID** of your device. The table in the lower area lists the netFIELD OS update images that are available for your device.

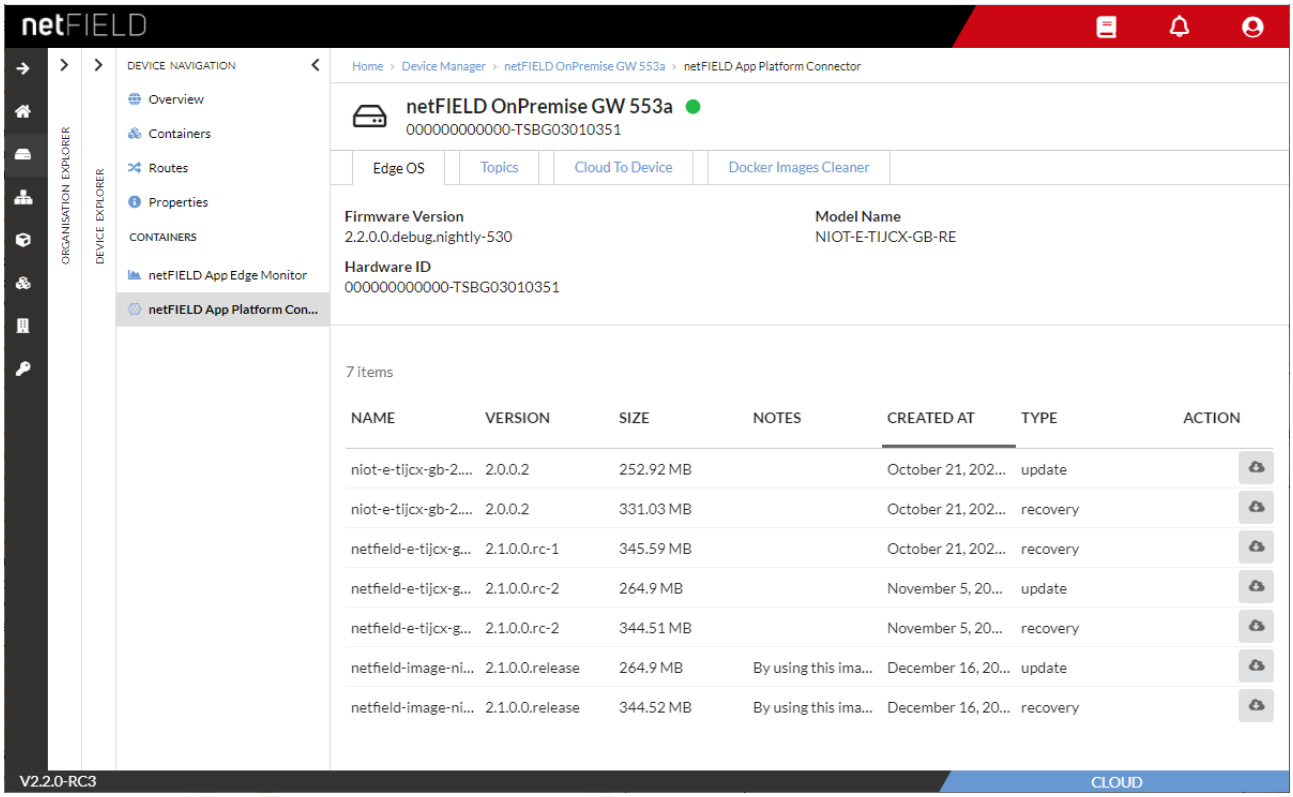

Figure 16: Platform Connector Edge OS tab

Element	Description
NAME	Name of the operating system image
VERSION	Version number of the image
SIZE	Size of the image file
NOTES	Additional information about the update/recovery process. Hover over the entry to display the full and unabridged text in an info box.
CREATED AT	Date and time of provision of the image in the netFIELD Portal (Note: Does not refer to the date of the actual "build" or "release" of the image).
TYPE	Type of the OS image: *Update* or *Recovery*. Note that netFIELD App Platform Connector supports only OS updates, not recoveries. (In a recovery, all individual containers and configuration settings – and thus the connection to the netFIELD Cloud – will be lost.)
ACTION	 Downloads and installs the OS update image on the device.
« ‹ 1 2 › »	If the list contains more than ten entries, you can scroll here to display the next ten items.

Table 6: Elements of Edge OS list

CAUTION

Risk of unsafe network operation caused by firmware update

Updating the operating system of your Edge Device will temporarily shut down the device and its normal communication functions. If your Edge Device is part of an OT network (like e.g. PROFINET), bear in mind that shutting down the Edge Device may also lead to a shutdown of the cyclic communication on the OT bus. Take precautions that shutting down the Edge Device and the OT bus will not endanger the safe operation of the plant.

- Click on the icon next to the image that you want to install on your device. Note that this must be a higher version than the one that is currently installed on your device. (A “lower” version can also be downloaded to the device, but its verification on the device will fail, resulting in a “Firmware installation failed” message.)
- In the **Deploy Edge OS Image** confirmation dialog window, click **Deploy** button to start the update process.

After having acknowledged a confirmation dialog, the image is downloaded to the device, then verified and installed. This may take a few minutes. The progress of the whole process is indicated by a progress bar:

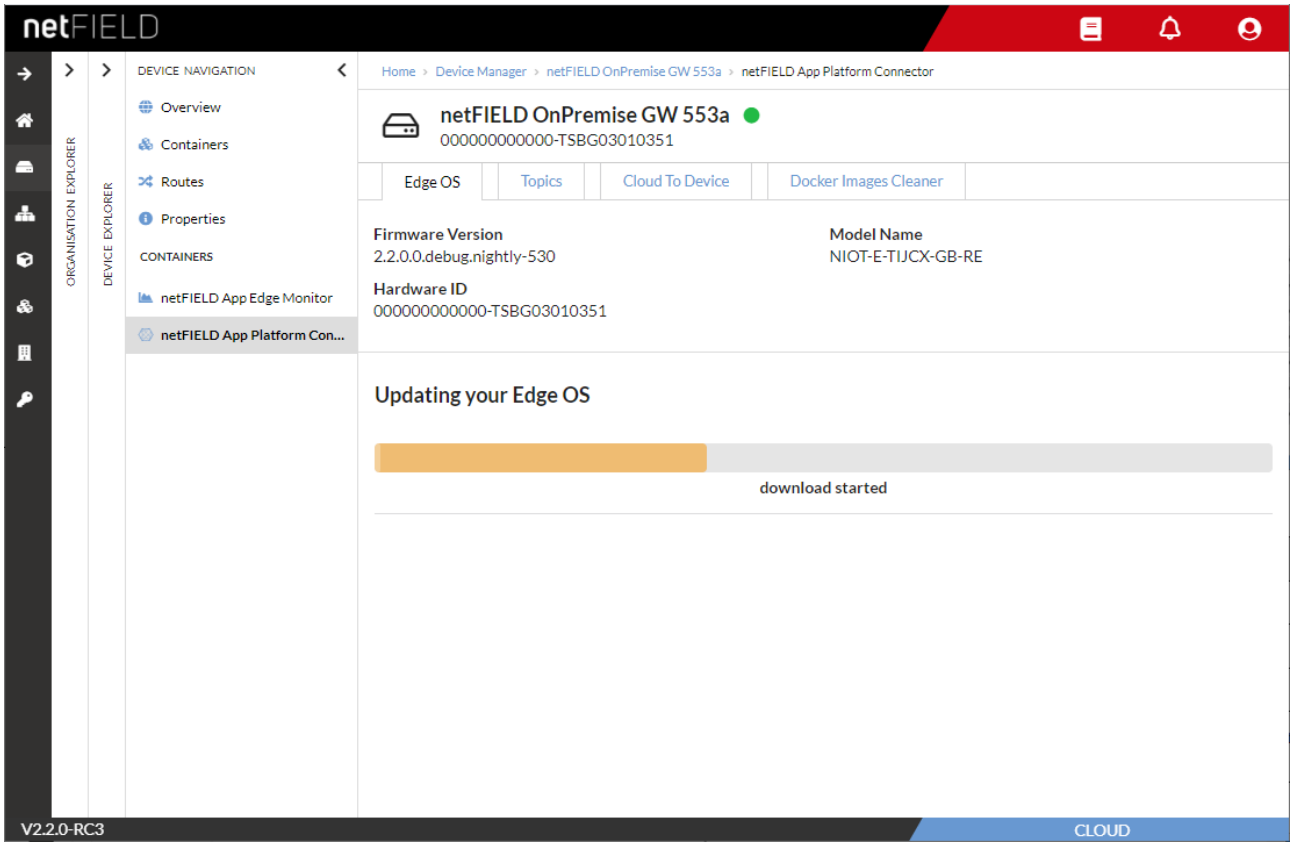

Figure 17: Update download started

After successful installation, the device is automatically restarted. This may take a few minutes.

4.2 Cloud-to-device tab

The netFIELD App Platform Connector allows you to publish arbitrary MQTT message data from the cloud dashboard (netFIELD Portal) to the MQTT Broker in your Edge Device.
Select your device in the Portal's Device Manager and open the **Cloud To Device** tab of the Platform Connector dashboard (DEVICE NAVIGATION > netFIELD App Platform Connector > Cloud To Device).

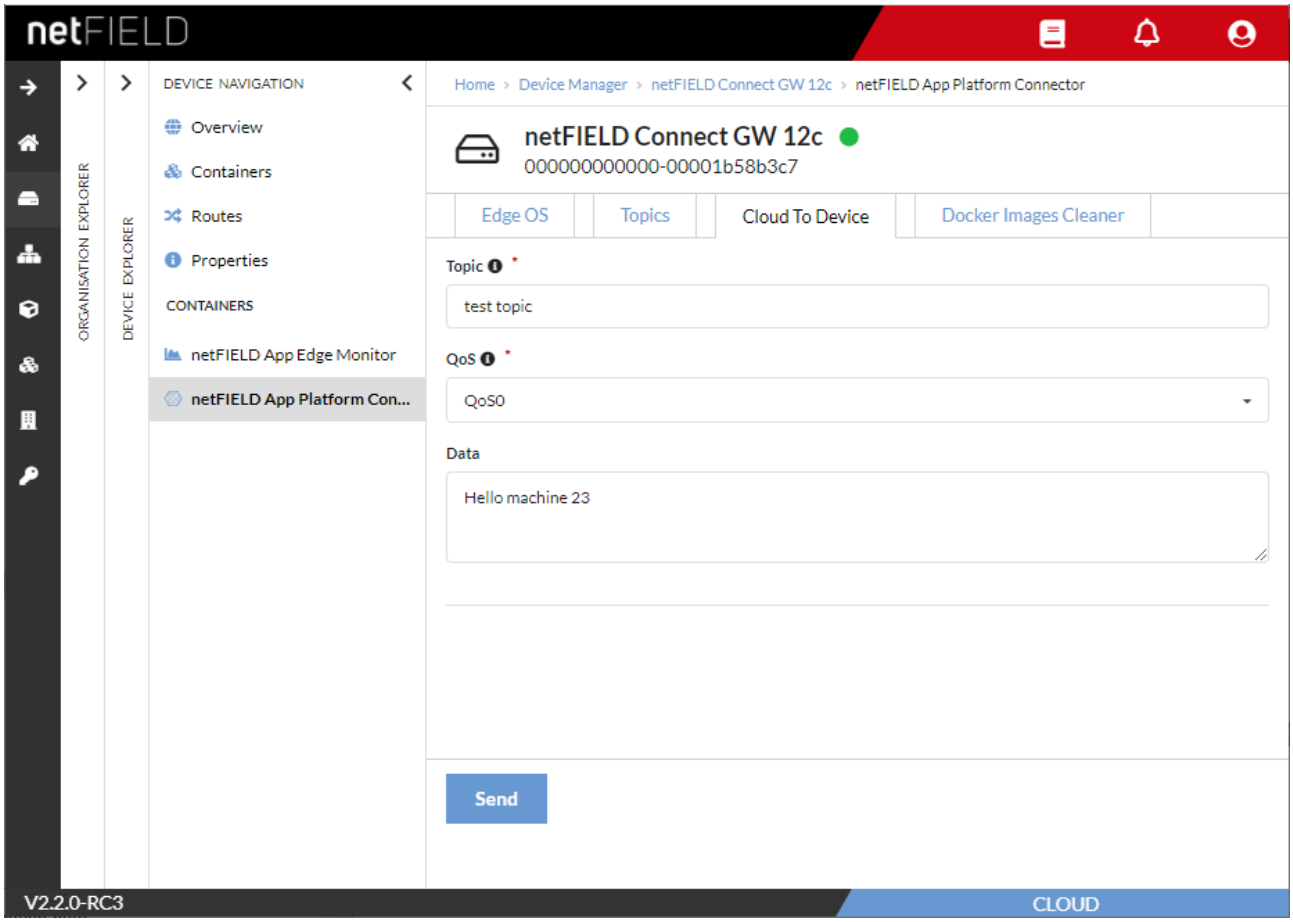

Figure 18: Cloud to device

- In the **Topic** field, enter the topic that you want to publish.
- In the **QoS** drop-down list, select the MQTT Quality of Service:
  QoS0: At most once
  QoS1: At least once
  QoS2: Exactly once
- In the **Data** field, enter the payload data that you want to publish. (Note that there are no guidelines concerning the payload data, it will be published as encoded string.)
- Click **Send** button.
- The topic and the message are transmitted from the Portal to the Platform Connector in the Edge Device.
  The Platform Connector (i.e. the MQTT client within the Platform Connector container) then sends a PUBLISH message for this topic to the MQTT Broker.

Note:

Once the broker has received the message, it is the responsibility of the broker to deliver the message to the subscribers. The Platform Connector will not get feedback about whether there are any subscribers or if any of the subscribing clients actually received the message from the broker.

4.3 Docker Images Cleaner tab

The netFIELD App Platform Connector allows you to delete obsolete container images in the **IoT Edge Docker** of the netFIELD OS of your device.

Note:
The **Delete Container** function for an installed container stops the container but does not erase its Docker image on the device. (See *Installed Containers* section in the *netFIELD Portal* manual, DOC190701OIxxEN) To erase these obsolete images (in order to save disk space), you can use the Docker Images Cleaner function of the Platform Connector app. It automatically cleans the **IoT Edge Docker** of all remaining images of deleted containers.

Select your device in the Portal's Device Manager and open the **Docker Images Cleaner** tab of the Platform Connector dashboard (DEVICE NAVIGATION > netFIELD App Platform Connector > Docker Images Cleaner).

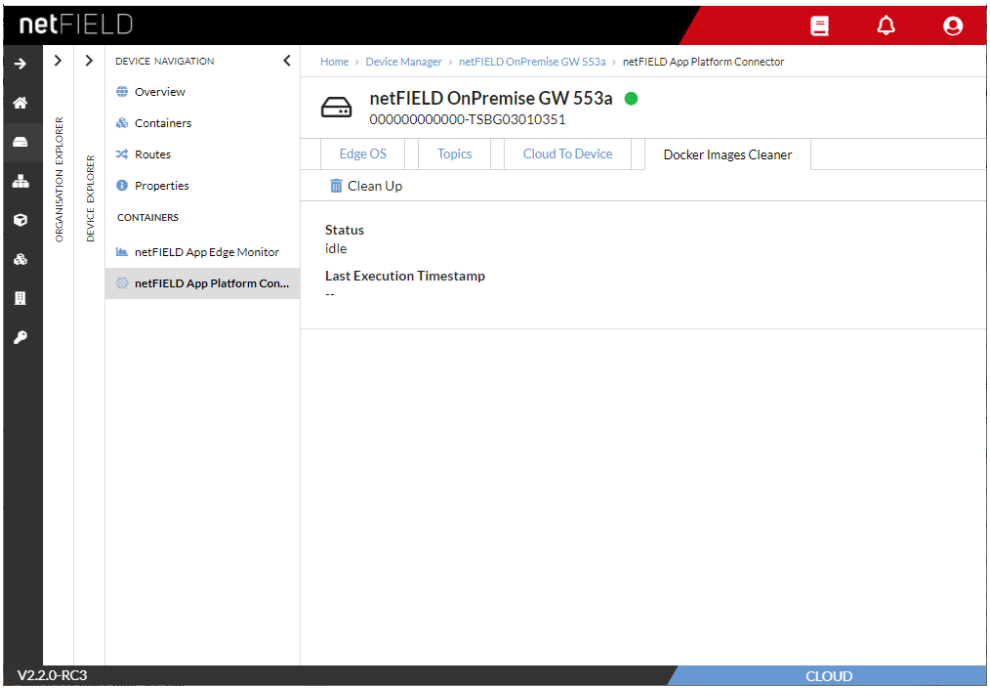

Figure 19: Docker Images Cleaner

➢ To delete unused container images in the IoT Edge Docker on your device, click **Clean Up** button.

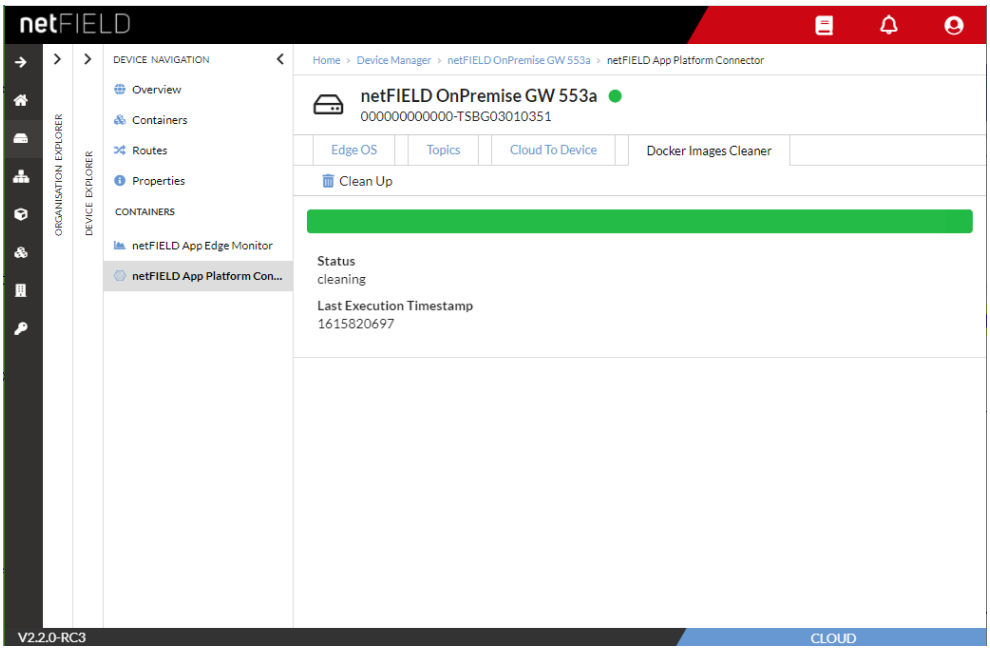

Figure 20: Cleaning

4.4 Device restart from cloud

The netFIELD App Platform Connector allows you to restart the netFIELD OS on your Edge Device (or virtual machine) from the cloud.
After deployment of the Platform Connector container on your device, a **Restart** button is added to the **Overview** page of your device in the Portal's Device Manager.
Be aware that this function only restarts the netFIELD OS on your device. It does not perform a full power cycle of your device's hardware.

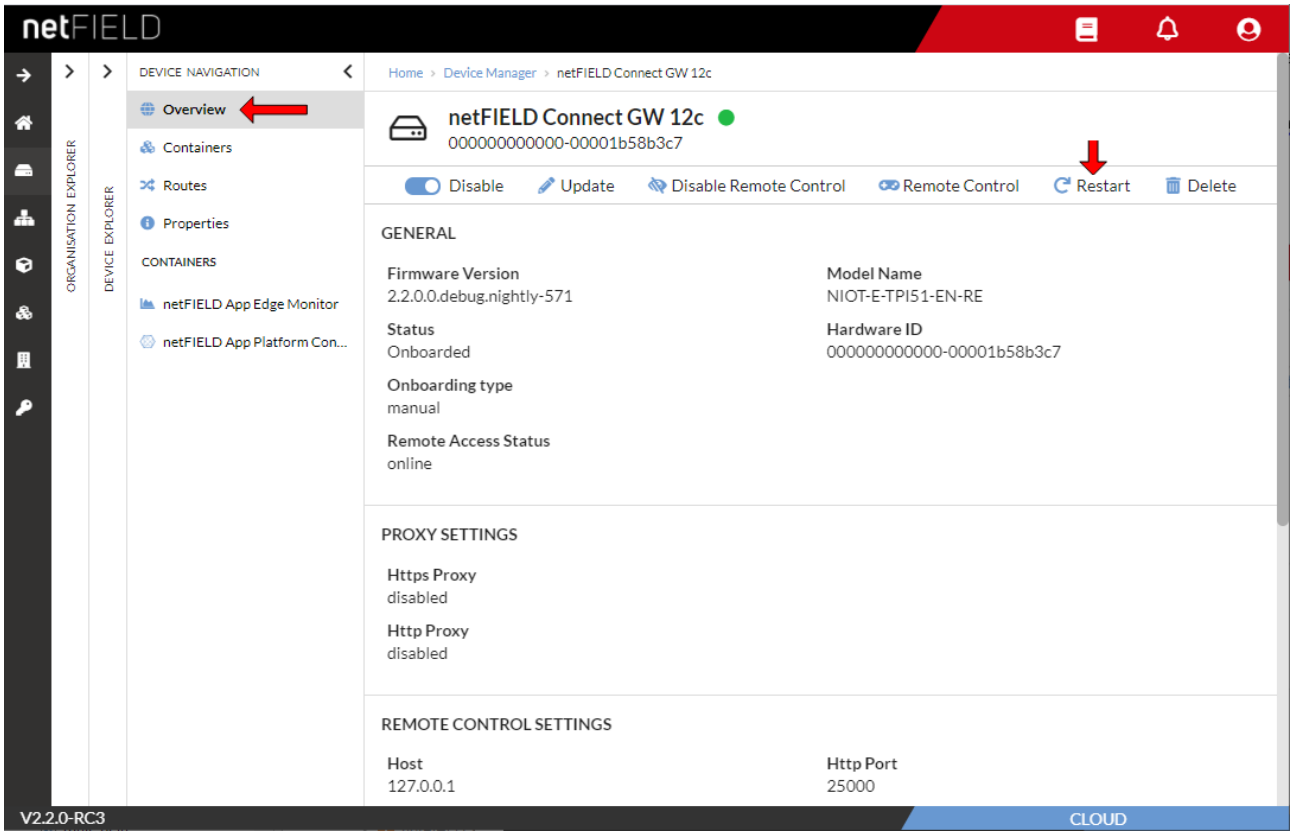

Figure 21: Restart button in Device Overview

➢ To restart the netFIELD OS on your device, select **Overview** in the **DEVICE NAVIGATION**, then click **Restart** button.

↬ The device disconnects from the cloud while the netFIELD OS is restarted, which is indicated by the dot changing from green (online) to red (offline). This might take a few minutes. After restart, the device will automatically reconnect again.

5 Legal notes

Copyright

Important notes

Utmost care was/is given in the preparation of the documentation at hand consisting of a user's manual, operating manual and any other document type and accompanying texts. However, errors cannot be ruled out. Therefore, we cannot assume any guarantee or legal responsibility for erroneous information or liability of any kind. You are hereby made aware that descriptions found in the user's manual, the accompanying texts and the documentation neither represent a guarantee nor any indication on proper use as stipulated in the agreement or a promised attribute. It cannot be ruled out that the user's manual, the accompanying texts and the documentation do not completely match the described attributes, standards or any other data for the delivered product. A warranty or guarantee with respect to the correctness or accuracy of the information is not assumed.

We reserve the right to modify our products and the specifications for such as well as the corresponding documentation in the form of a user's manual, operating manual and/or any other document types and accompanying texts at any time and without notice without being required to notify of said modification. Changes shall be taken into account in future manuals and do not represent an obligation of any kind, in particular there shall be no right to have delivered documents revised. The manual delivered with the product shall apply.

Under no circumstances shall Hilscher Gesellschaft für Systemautomation mbH be liable for direct, indirect, ancillary or subsequent damage, or for any loss of income, which may arise after use of the information contained herein.

Liability disclaimer

The hardware and/or software was created and tested by Hilscher Gesellschaft für Systemautomation mbH with utmost care and is made available as is. No warranty can be assumed for the performance or flawlessness of the hardware and/or software under all application conditions and scenarios and the work results achieved by the user when using the hardware and/or software. Liability for any damage that may have occurred as a result of using the hardware and/or software or the corresponding documents shall be limited to an event involving willful intent or a grossly negligent violation of a fundamental contractual obligation. However, the right to assert damages due to a violation of a fundamental contractual obligation shall be limited to contract-typical foreseeable damage.

It is hereby expressly agreed upon in particular that any use or utilization of the hardware and/or software in connection with

- Flight control systems in aviation and aerospace;
- Nuclear fission processes in nuclear power plants;
- Medical devices used for life support and
- Vehicle control systems used in passenger transport

shall be excluded. Use of the hardware and/or software in any of the following areas is strictly prohibited:

- For military purposes or in weaponry;
- For designing, engineering, maintaining or operating nuclear systems;
- In flight safety systems, aviation and flight telecommunications systems;
- In life-support systems;
- In systems in which any malfunction in the hardware and/or software may result in physical injuries or fatalities.

You are hereby made aware that the hardware and/or software was not created for use in hazardous environments, which require fail-safe control mechanisms. Use of the hardware and/or software in this kind of environment shall be at your own risk; any liability for damage or loss due to impermissible use shall be excluded.

Warranty

Hilscher Gesellschaft für Systemautomation mbH hereby guarantees that the software shall run without errors in accordance with the requirements listed in the specifications and that there were no defects on the date of acceptance. The warranty period shall be 12 months commencing as of the date of acceptance or purchase (with express declaration or implied, by customer's conclusive behavior, e.g. putting into operation permanently).

The warranty obligation for equipment (hardware) we produce is 36 months, calculated as of the date of delivery ex works. The aforementioned provisions shall not apply if longer warranty periods are mandatory by law pursuant to Section 438 (1.2) BGB, Section 479 (1) BGB and Section 634a (1) BGB [Bürgerliches Gesetzbuch; German Civil Code] If, despite of all due care taken, the delivered product should have a defect, which already existed at the time of the transfer of risk, it shall be at our discretion to either repair the product or to deliver a replacement product, subject to timely notification of defect.

The warranty obligation shall not apply if the notification of defect is not asserted promptly, if the purchaser or third party has tampered with the products, if the defect is the result of natural wear, was caused by unfavorable operating conditions or is due to violations against our operating regulations or against rules of good electrical engineering practice, or if our request to return the defective object is not promptly complied with.

Costs of support, maintenance, customization and product care

Please be advised that any subsequent improvement shall only be free of charge if a defect is found. Any form of technical support, maintenance and customization is not a warranty service, but instead shall be charged extra.

Additional guarantees

Although the hardware and software was developed and tested in-depth with greatest care, Hilscher Gesellschaft für Systemautomation mbH shall not assume any guarantee for the suitability thereof for any purpose that was not confirmed in writing. No guarantee can be granted whereby the hardware and software satisfies your requirements, or the use of the hardware and/or software is uninterruptable or the hardware and/or software is fault-free.

It cannot be guaranteed that patents and/or ownership privileges have not been infringed upon or violated or that the products are free from third-party influence. No additional guarantees or promises shall be made as to whether the product is market current, free from deficiency in title, or can be integrated or is usable for specific purposes, unless such guarantees or promises are required under existing law and cannot be restricted.

Confidentiality

The customer hereby expressly acknowledges that this document contains trade secrets, information protected by copyright and other patent and ownership privileges as well as any related rights of Hilscher Gesellschaft für Systemautomation mbH. The customer agrees to treat as confidential all of the information made available to customer by Hilscher Gesellschaft für Systemautomation mbH and rights, which were disclosed by Hilscher Gesellschaft für Systemautomation mbH and that were made accessible as well as the terms and conditions of this agreement itself.

The parties hereby agree to one another that the information that each party receives from the other party respectively is and shall remain the intellectual property of said other party, unless provided for otherwise in a contractual agreement.

The customer must not allow any third party to become knowledgeable of this expertise and shall only provide knowledge thereof to authorized users as appropriate and necessary. Companies associated with the customer shall not be deemed third parties. The customer must obligate authorized users to confidentiality. The customer should only use the confidential information in connection with the performances specified in this agreement.

The customer must not use this confidential information to his own advantage or for his own purposes or rather to the advantage or for the purpose of a third party, nor must it be used for commercial purposes and this confidential information must only be used to the extent provided for in this agreement or otherwise to the extent as expressly authorized by the disclosing party in written form. The customer has the right, subject to the obligation to confidentiality, to disclose the terms and conditions of this agreement directly to his legal and financial consultants as would be required for the customer's normal business operation.

Export provisions

The delivered product (including technical data) is subject to the legal export and/or import laws as well as any associated regulations of various countries, especially such laws applicable in Germany and in the United States. The products / hardware / software must not be exported into such countries for which export is prohibited under US American export control laws and its supplementary provisions. You hereby agree to strictly follow the regulations and to yourself be responsible for observing them. You are hereby made aware that you may be required to obtain governmental approval to export, reexport or import the product.

List of Figures

List of Tables

Contacts

HEADQUARTERS

Germany

Hilscher Gesellschaft für
Systemautomation mbH
Rheinstrasse 15
65795 Hattersheim
Phone: +49 (0) 6190 9907-0
Fax: +49 (0) 6190 9907-50
E-mail: info@hilscher.com

Support

Phone: +49 (0) 6190 9907-99
E-mail: de.support@hilscher.com

SUBSIDIARIES

China

Hilscher Systemautomation (Shanghai) Co. Ltd.
200010 Shanghai
Phone: +86 (0) 21-6355-5161
E-mail: info@hilscher.cn

Support

Phone: +86 (0) 21-6355-5161
E-mail: cn.support@hilscher.com

France

Hilscher France S.a.r.l.
69500 Bron
Phone: +33 (0) 4 72 37 98 40
E-mail: info@hilscher.fr

Support

Phone: +33 (0) 4 72 37 98 40
E-mail: fr.support@hilscher.com

India

Hilscher India Pvt. Ltd.
Pune, Delhi, Mumbai
Phone: +91 8888 750 777
E-mail: info@hilscher.in

Italy

Hilscher Italia S.r.l.
20090 Vimodrone (MI)
Phone: +39 02 25007068
E-mail: info@hilscher.it

Support

Phone: +39 02 25007068
E-mail: it.support@hilscher.com

Japan

Hilscher Japan KK
Tokyo, 160-0022
Phone: +81 (0) 3-5362-0521
E-mail: info@hilscher.jp

Support

Phone: +81 (0) 3-5362-0521
E-mail: jp.support@hilscher.com

Korea

Hilscher Korea Inc.
Seongnam, Gyeonggi, 463-400
Phone: +82 (0) 31-789-3715
E-mail: info@hilscher.kr

Switzerland

Hilscher Swiss GmbH
4500 Solothurn
Phone: +41 (0) 32 623 6633
E-mail: info@hilscher.ch

Support

Phone: +49 (0) 6190 9907-99
E-mail: ch.support@hilscher.com

USA

Hilscher North America, Inc.
Lisle, IL 60532
Phone: +1 630-505-5301
E-mail: info@hilscher.us

Support

Phone: +1 630-505-5301
E-mail: us.support@hilscher.com