RAK10701-P Field Tester Pro for LoRaWAN Quick Start Guide

Prerequisites

What Do You Need?

Before going through each and every step in the installation guide of the RAK10701-P Field Tester Pro for LoRaWAN, make sure to prepare the necessary items listed below:

Hardware Tools

1. RAK10701-P Field Tester Pro for LoRaWAN
2. LoRa SubGhz Antenna with RP-SMA connector
3. USB Type-C Cable
4. Windows/Linux/macOS for PC or iOS/Android for mobile

Software Tools

WisToolBox for configuration and firmware update.

> 📝 **NOTE:**
>
> It is mandatory that you are within the coverage of the LoRaWAN Gateway of the network you are trying to join. Without the coverage, the Field Tester will not be useable.

Product Configuration

RAK10701-P Field Tester Pro Physical Interface

The user interface of the RAK10701-P Field Tester Pro for LoRaWAN is via TFT Touchscreen LCD and one pushbutton at the side. There is also an external LoRA antenna port via RP-SMA connector and USB-C port for charging and configuration if connected to a PC.

Figure 1: Parts of RAK10701-P

> 📝 **NOTE:**
>
> You have to ensure that the LoRa antenna is attached before turning on the device.

1. To turn on the device, you have to press and hold the button for at least five seconds.

Figure 2: RAK10701-P button to turn on

> 📝 **NOTE:**
>
> The same button can be used to power off. You have to hold it as well for at least five seconds.

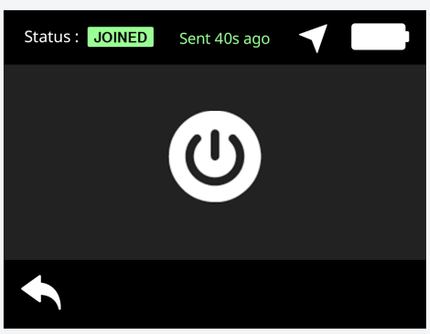

Figure 3: RAK10701-P power off

2. When the device initializes, it will show the RAK logo on the screen. If there is any initialization error, it will be shown on the upper right section of the screen as well. A properly working device should not have any errors shown.

Figure 4: RAK10701-P power up successful

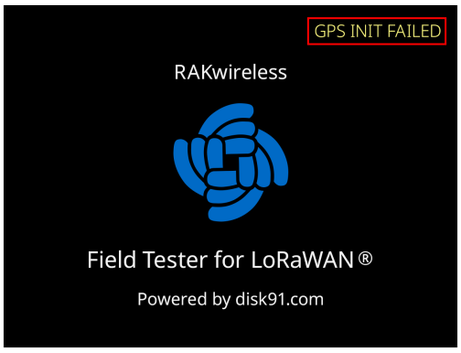

Figure 5: GPS error on boot up sequence

3. After the successful boot-up, the main home screen will be shown. Take note, that there will be no data at the first start of the device.

> 📝 **NOTE:**
>
> The field tester must be outside and has a clear view of the sky to get GPS coordinates. The GPS antenna is located on top of the device beside the RP-SMA connector of the LoRa Antenna.
>
> If you are indoors, there will be no reception of the GPS signal. The latitude and longitude data will be empty.

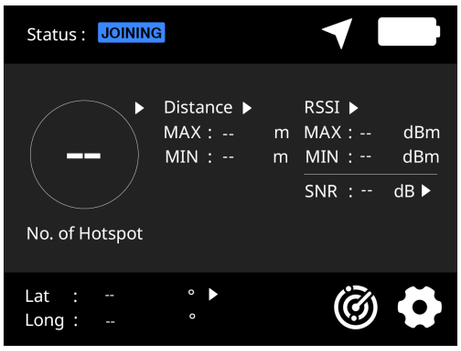

Figure 6: RAK10701-P Main Page waiting for valid data

4. Once fully powered on, the external button at the side can sleep or wake up the display on the LCD screen via a single press on it.

5. If the device is connected via USB-C to a computer, then the button is pressed, it will not remove the display but will lock the screen (touch screen behavior is disabled).

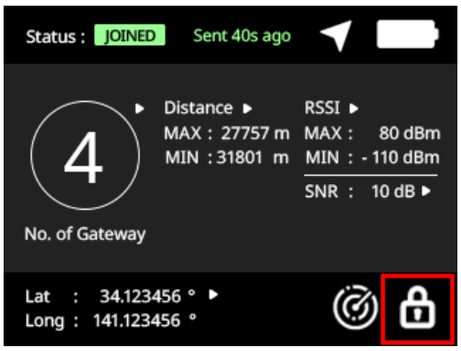

Figure 7: RAK10701-P locked screen

LoRaWAN Network Servers Guide for RAK10701-P Field Tester Pro

The field tester supports different network servers and can be used as well on others not listed in this guide as long as the uplink and downlink packets are configured correctly.

You can check each guide on how to use the RAK10701-P Field Tester Pro for LoRaWAN in the following network servers.

- Helium
- The Things Network
- Chirpstack (with Datacake)
- Loriot (with Datacake)

Additional information:

- Packet Frame Format

> 📝 **NOTE:**
>
> This section will focus on the configuration of each network server. The procedure of Device Configuration of RAK10701-P via WisToolBox is the same for all network server and will be covered in a separate section of the guide.

RAK10701-P Field Tester Pro Guide for the Helium Network

RAK10701-P can be manually registered to Helium Console . This is a public LoRaWAN network server that you can use for your LoRaWAN end-devices powered by community-driven Helium Hotspots. This guide will show every detail of how to prepare the Helium Console for your RAK10701-P Field Tester Pro.

> 📝 **NOTE:**
>
> This guide is based on disk19 guide for the Field Tester configuration for the Helium Console.

1. You need to register an account and then purchase data credits (DC) to use the network. If you are a new user, there are free data credits (DC) included in your new account to get you started quickly.

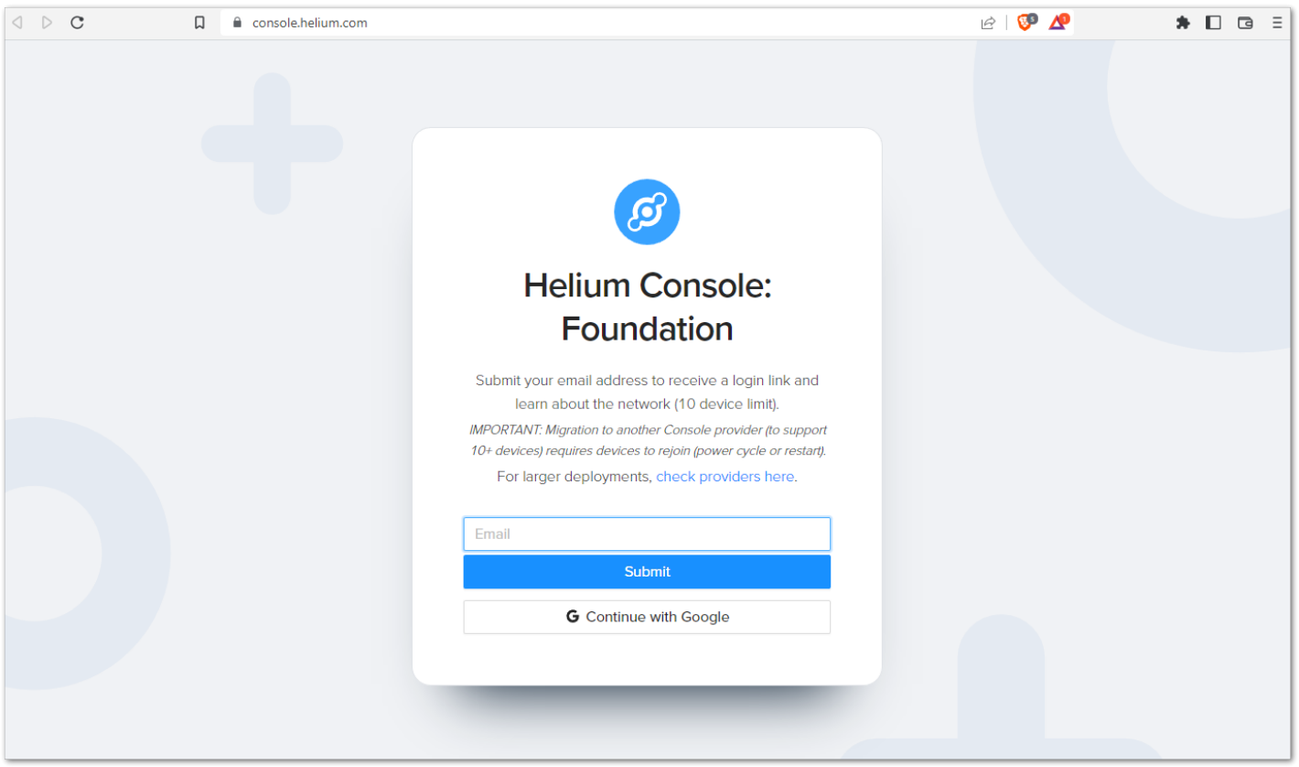

Figure 8: Heluim Console

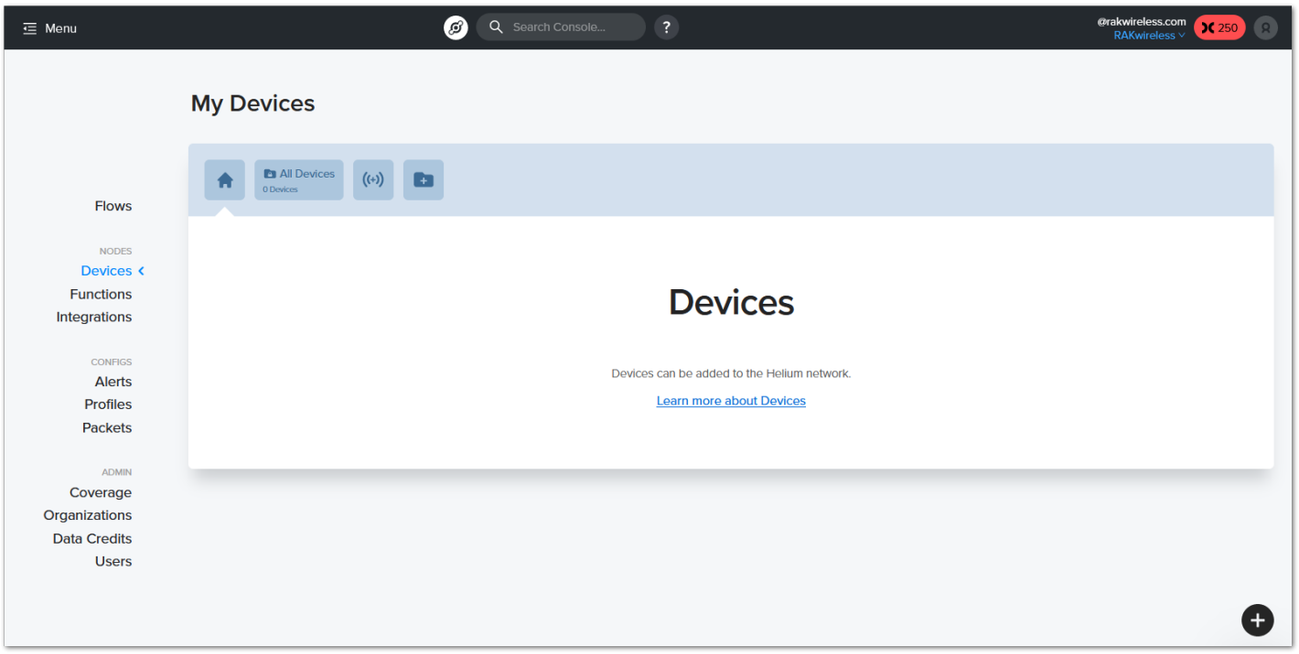

Figure 9: Console Home Page

2. Once you are logged in, you can start adding your device. You have two ways to add a device as shown in the image.

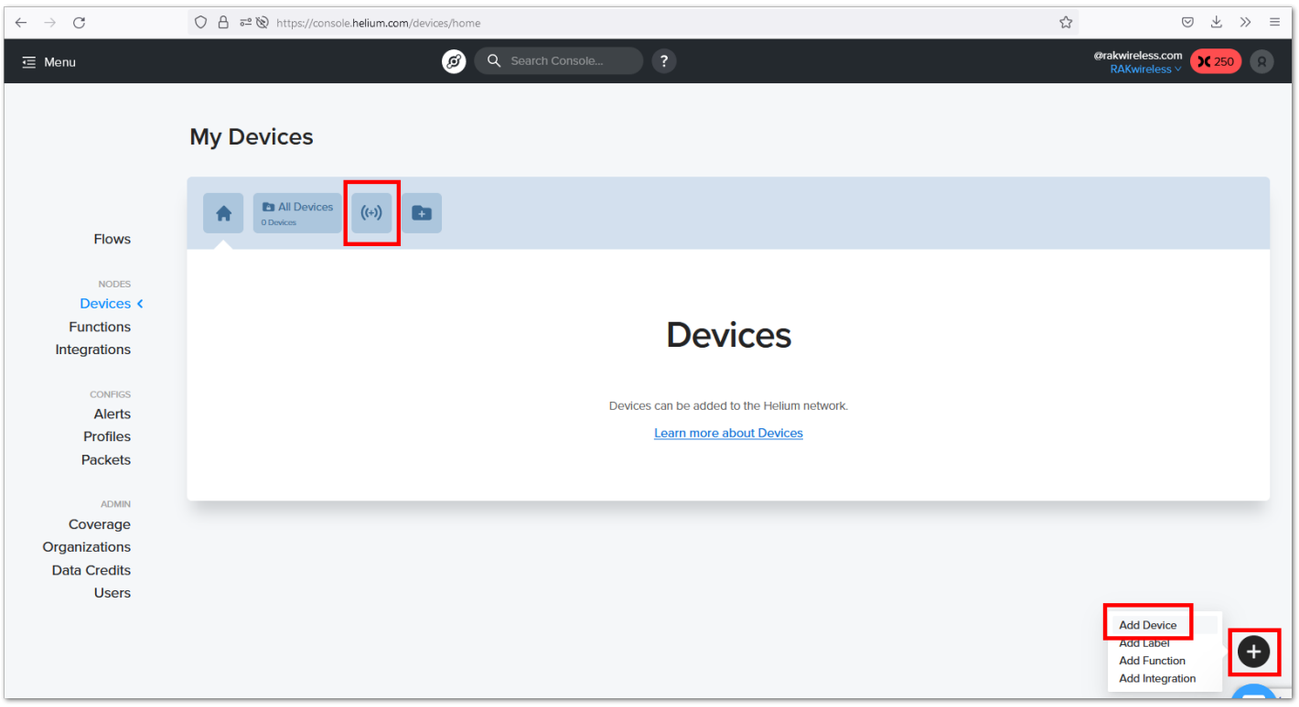

Figure 10: Adding Device

3. The newly added device parameters will be shown. You have to put a device name and click save.

> 📝 **NOTE:**
>
> The DEVEUI, APPEUI, and APPKEY are important in this step. These values must be configured on your RAK10701-P device using WisToolBox which will be covered later in this guide.

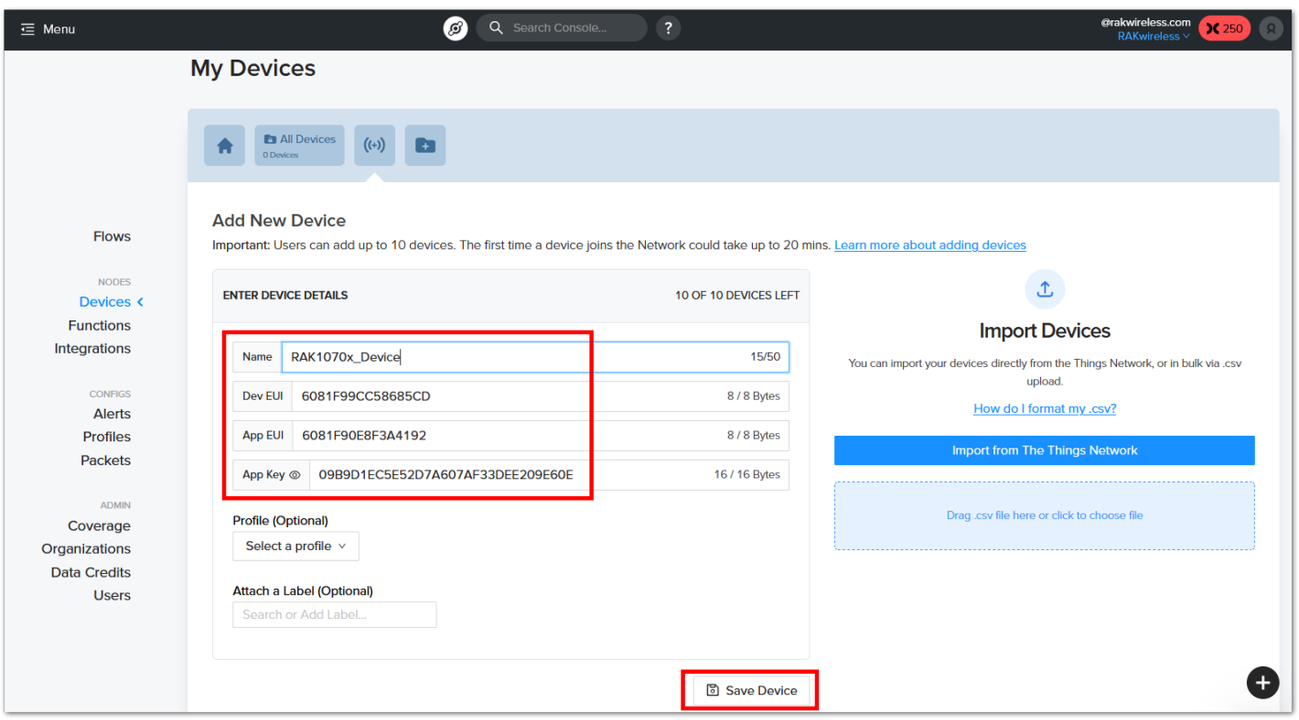

Figure 11: Configuring Device Name

4. The device will be added to the blockchain and it will show pending beside its name.

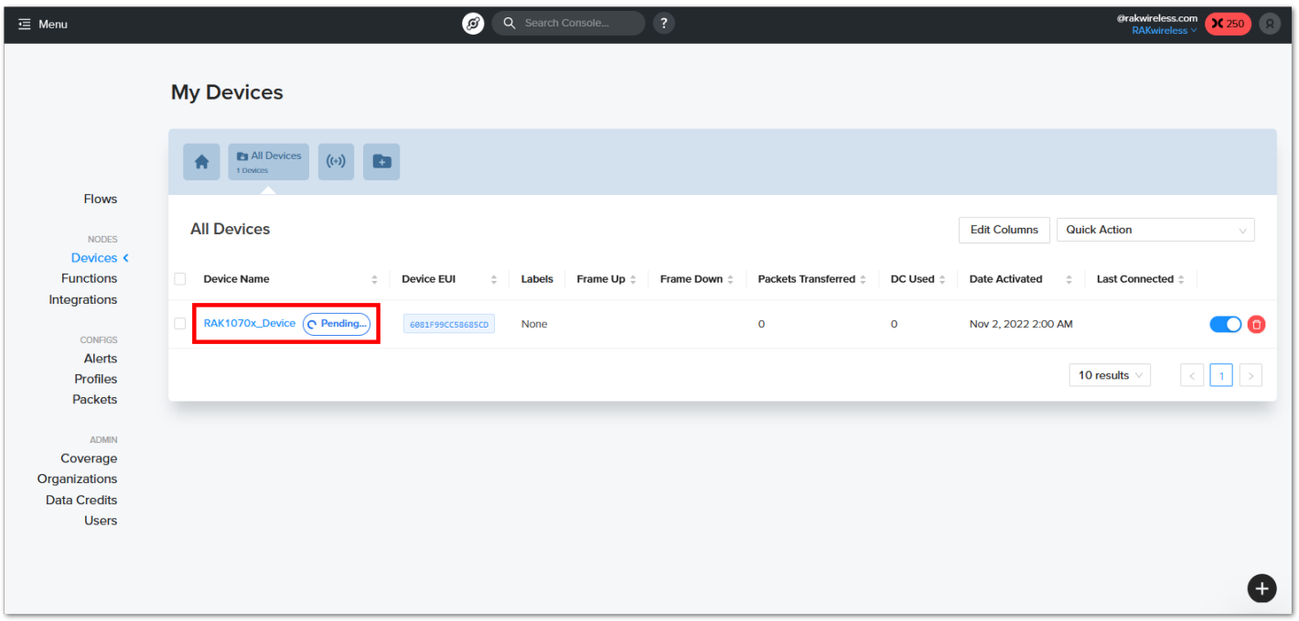

Figure 12: Pending Device Status

5. While waiting for the device to be added to the blockchain, you can create a `Label`. This will allow you to group your device to have a common setting. This will be needed to attach the needed integrations to the backend server of `dev.disk91.com`. You have to click the folder with the + icon and add a `Label` name then click `Save Label`. The newly created label should now be shown in the `Devices` console.

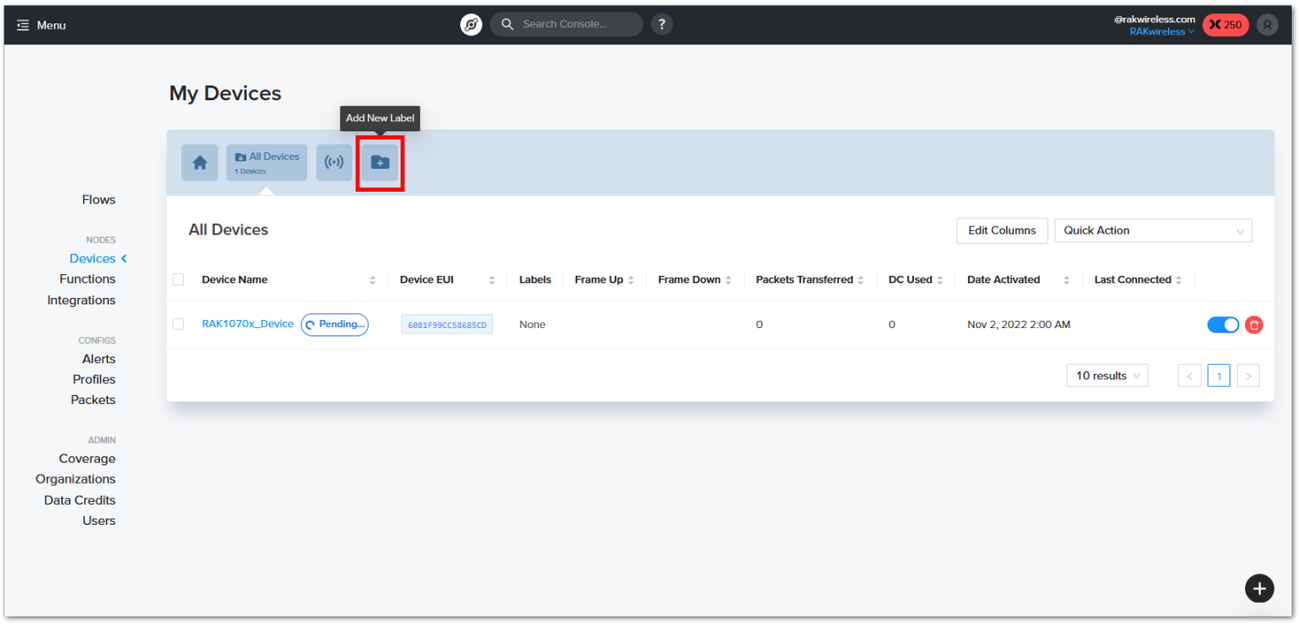

Figure 13: Add Label icon

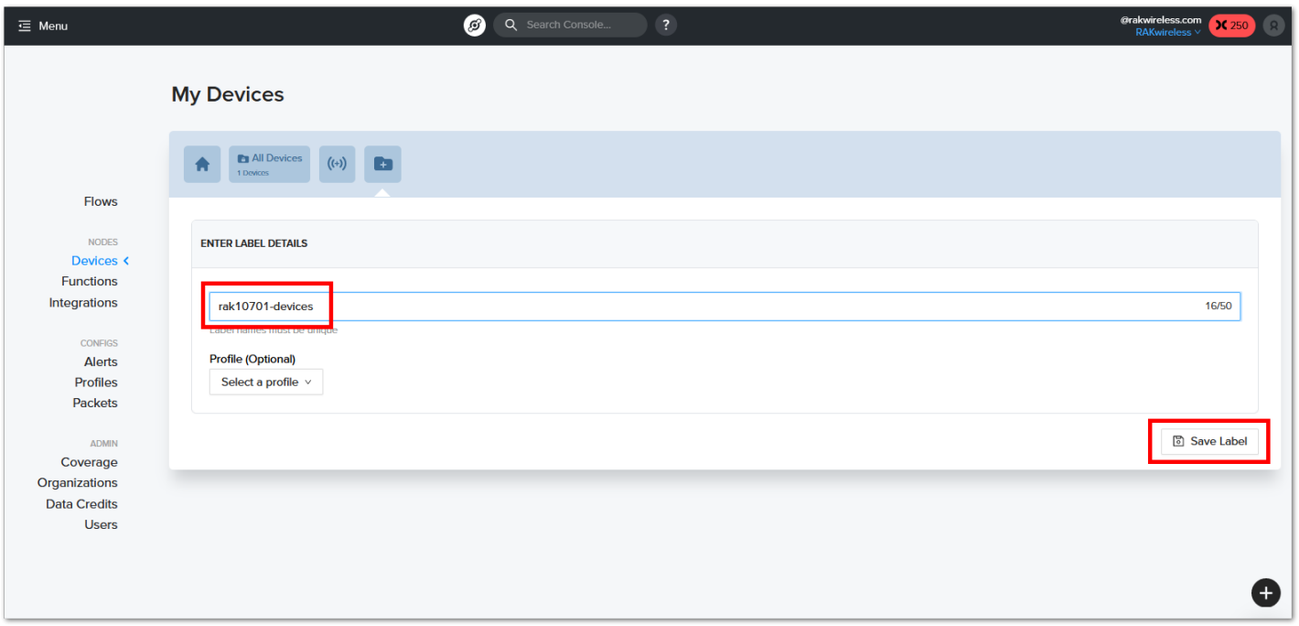

Figure 14: Add Label Name

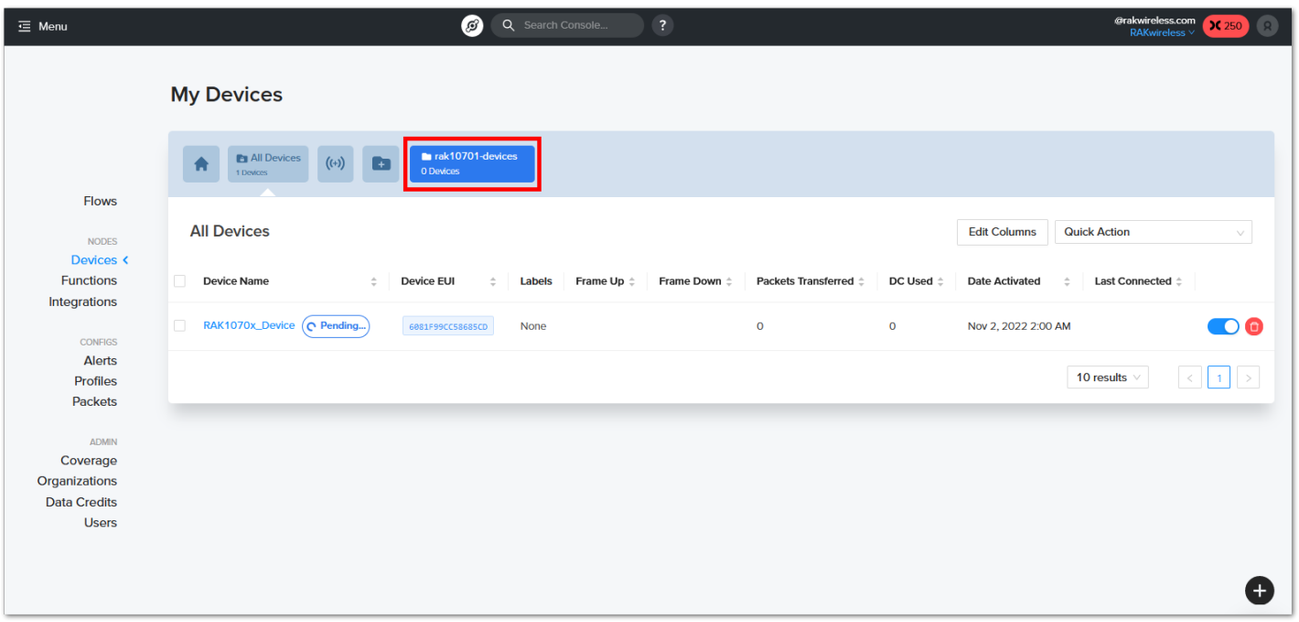

Figure 15: Label created successfully

6. Once the `Label` is created you have to associate it on the RAK10701 device. You can attach the `Label` on the device by clicking the **Add Label** button.

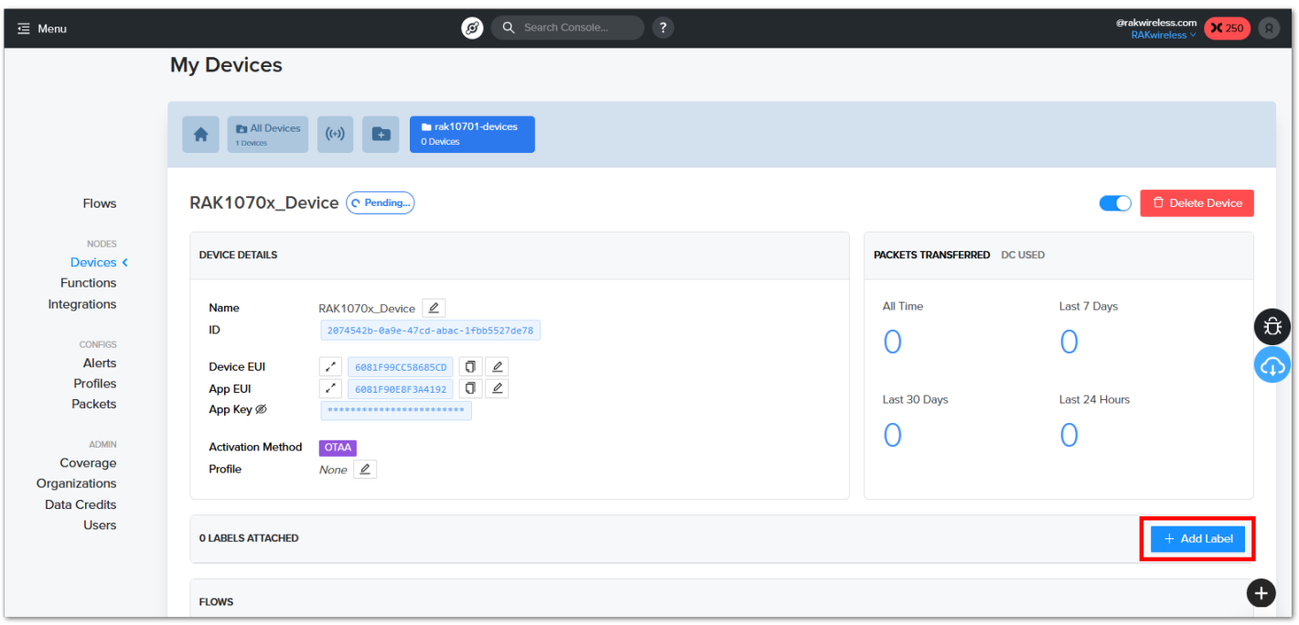

Figure 16: Attach a label to the device

7. A pop-up will be shown and you have to select the correct `Label` created for RAK10701 then click **Add Label**.

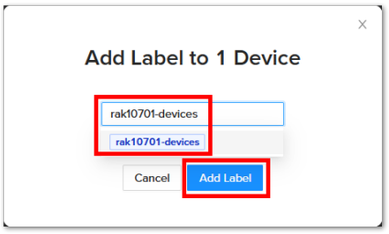

Figure 17: Drop-down on label selection

8. After successful attachment of `Label` on the devices, it should show one (1) device is under that `Label`. The device is properly labeled which is needed for the next steps - `Integrations` and `Flow`.

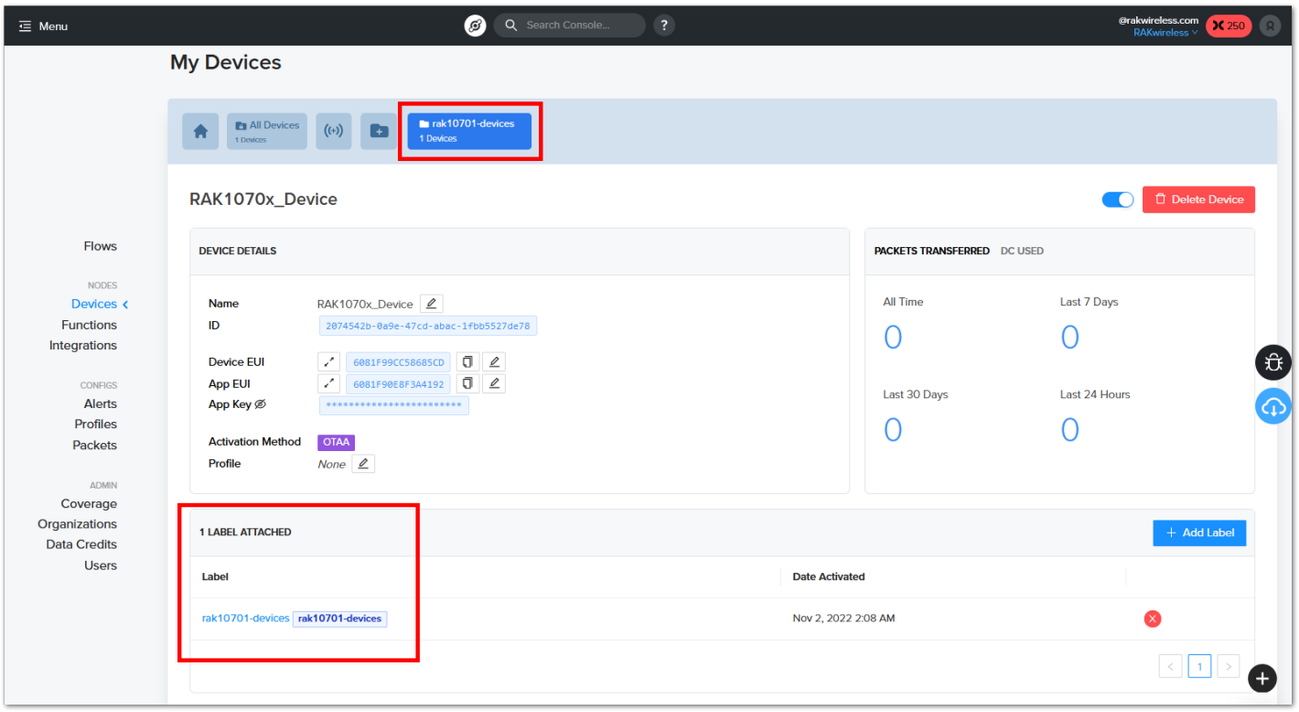

Figure 18: Label added on the RAK10701 device

9. To connect the backend server, you have to create an `Integration`.

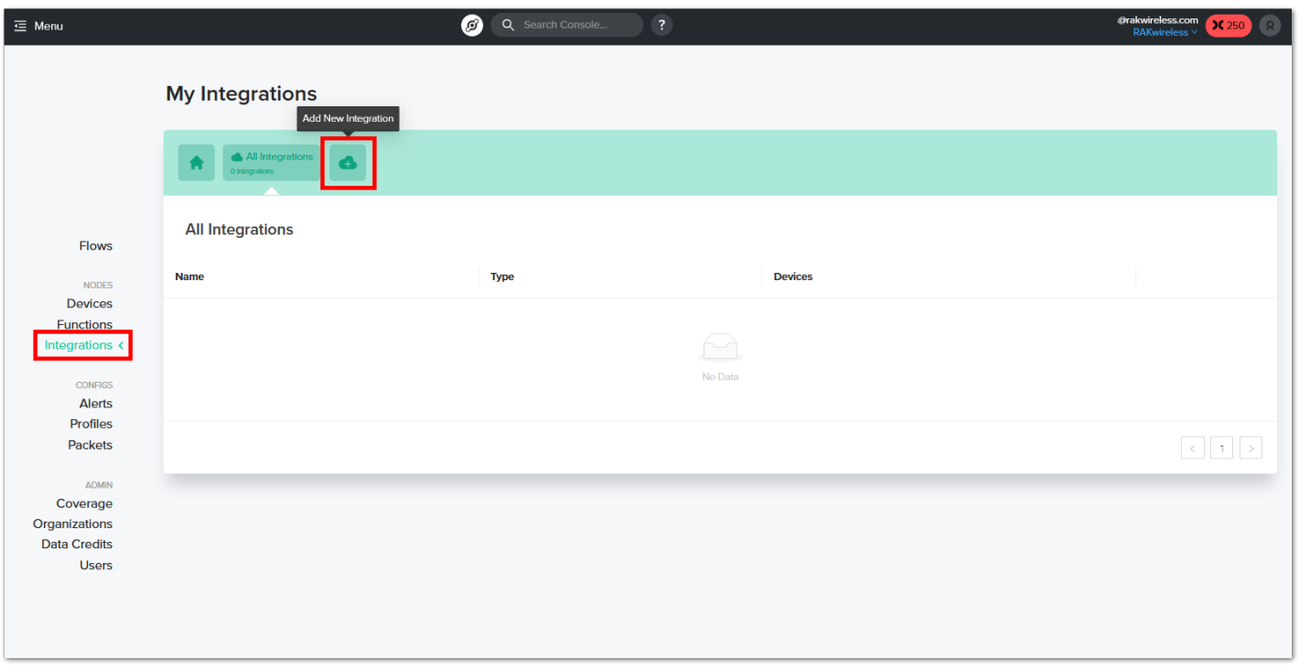

Figure 19: Add integration

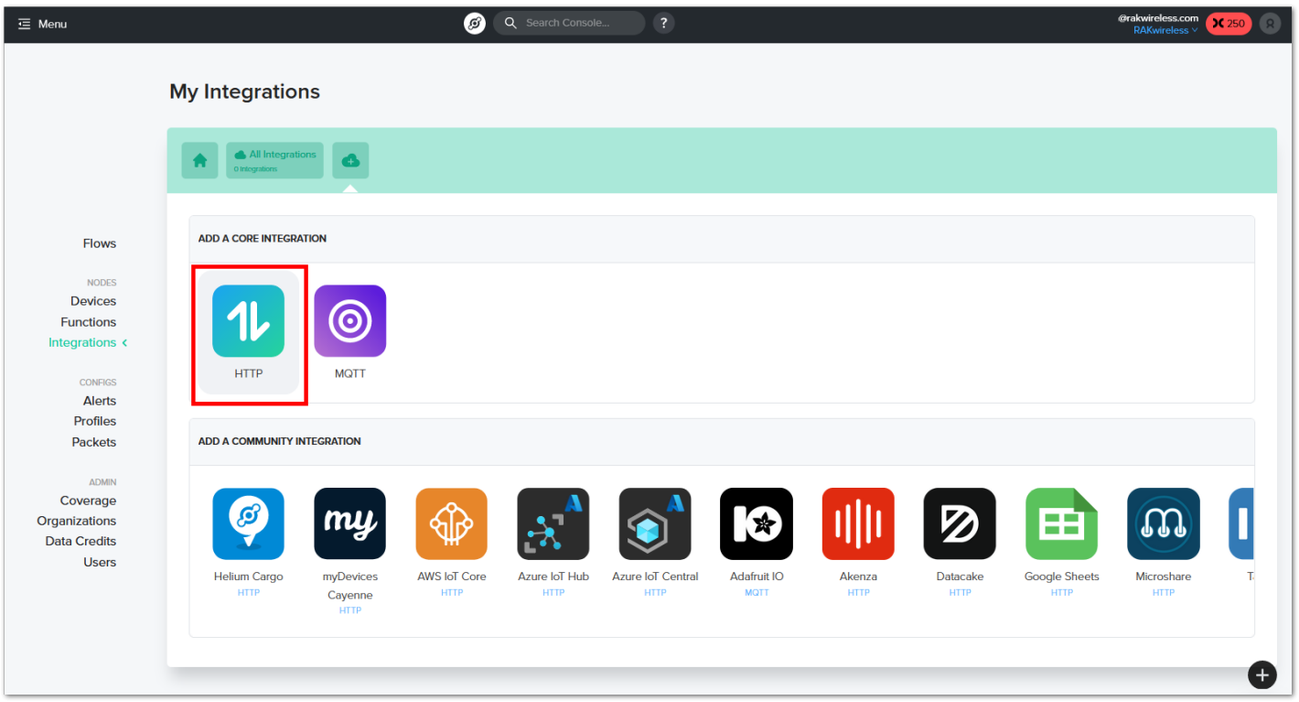

Figure 20: HTTP integration

10. Then you must proceed to steps 2 and 3 sections of the `Integration` settings. You have to select `POST` then on the Endpoint URL, you must put `https://dev.disk91.com/fieldtester/helium/v3`. It is also needed to put the integration name before the clicking Add integration button.

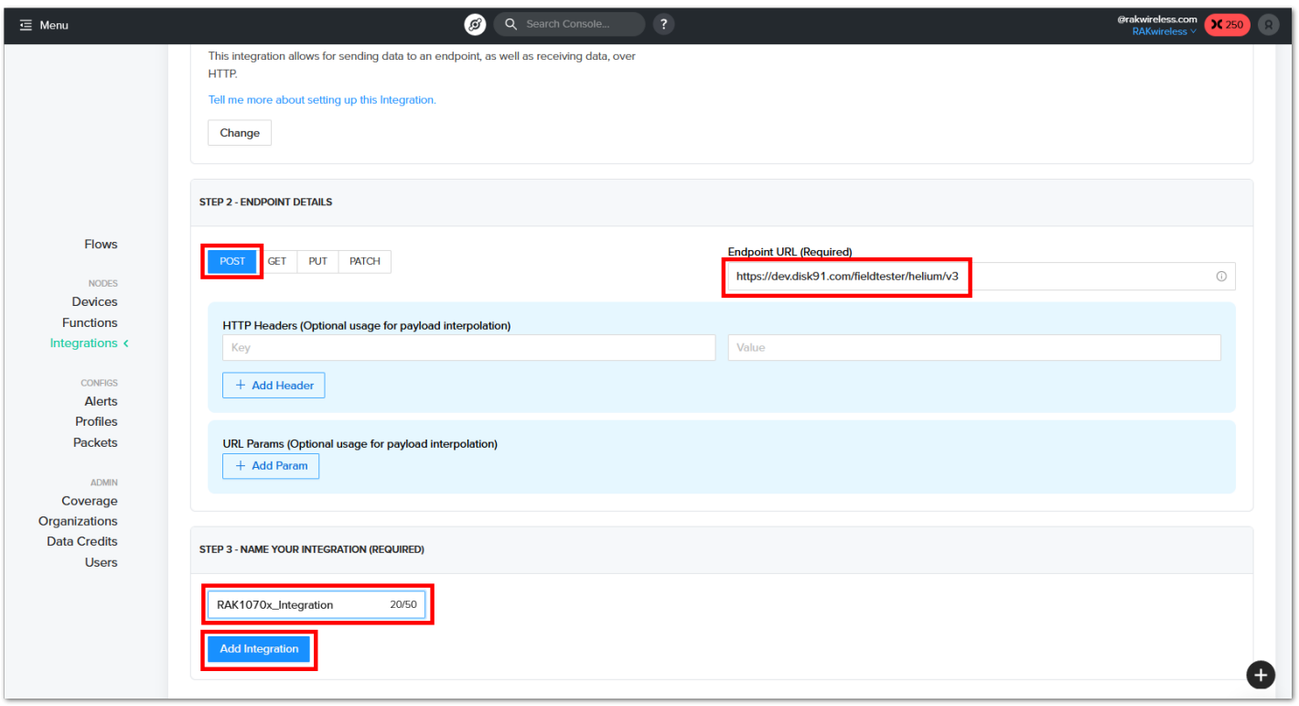

Figure 21: Details of HTTP Integration

11. After preparing the device and the integration, you can now proceed with creating the flow to connect them. You have to click `Flows` and then the `+` icon on `NODES`.

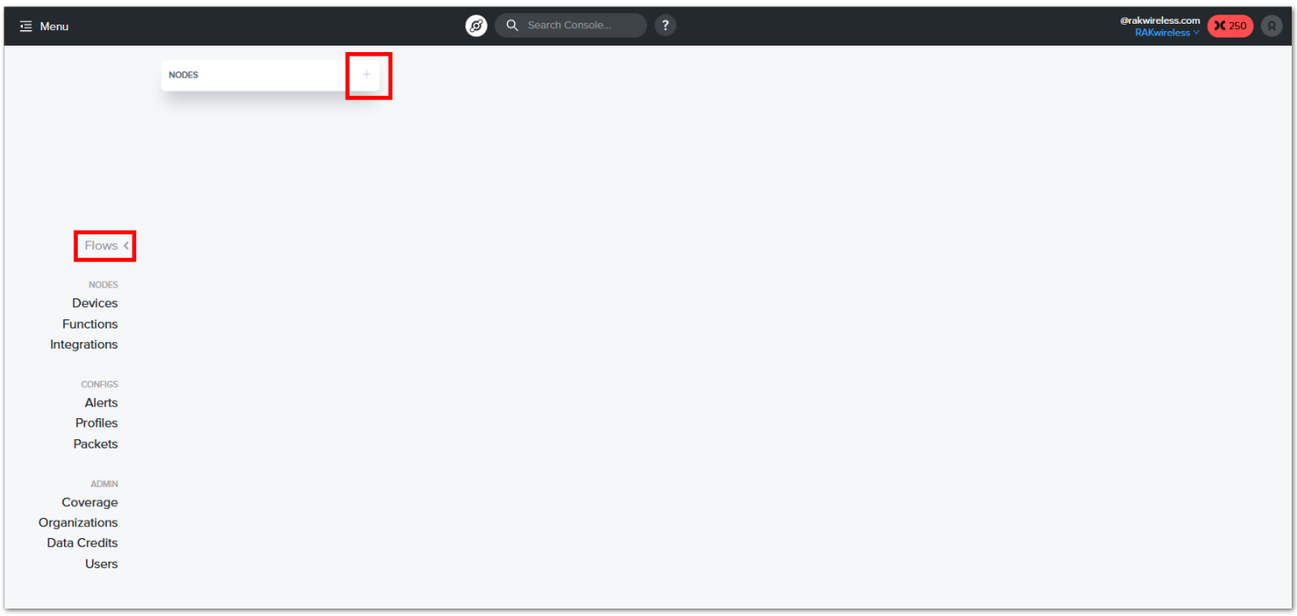

Figure 22: Setting up the Flows

12. You must select `Labels` and `Integrations` and then drag the correct blocks on the flows canvas.

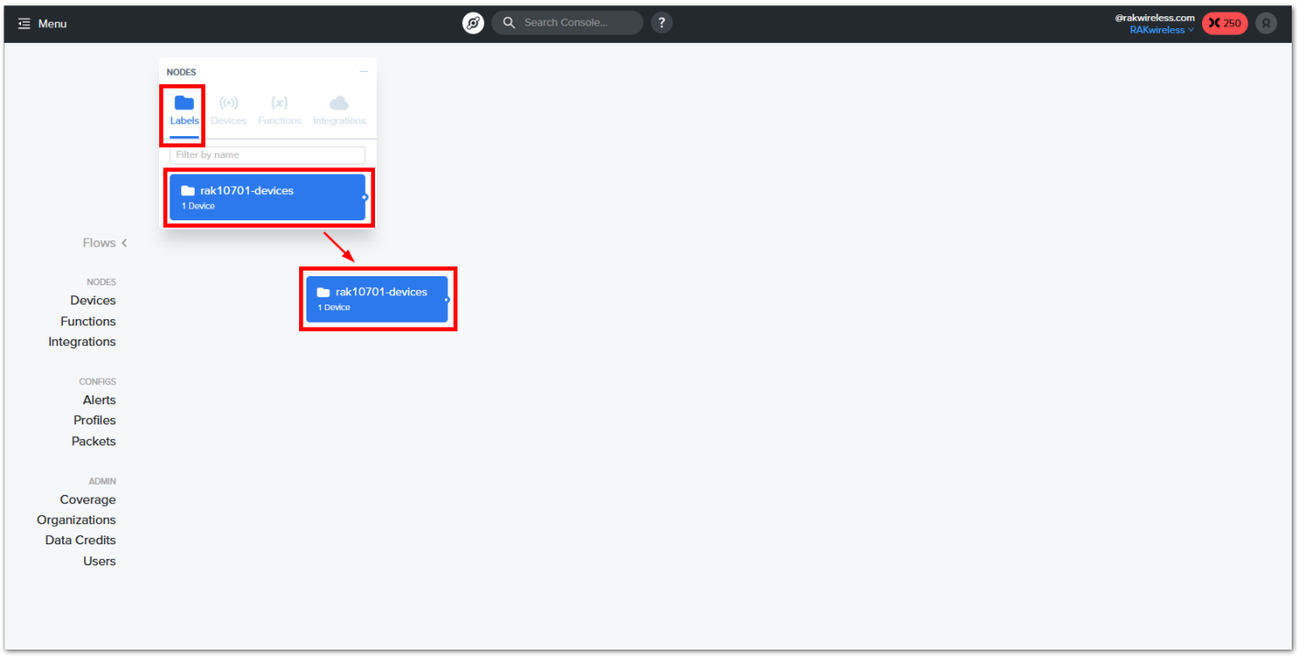

Figure 23: Drag the rak10701-devices label

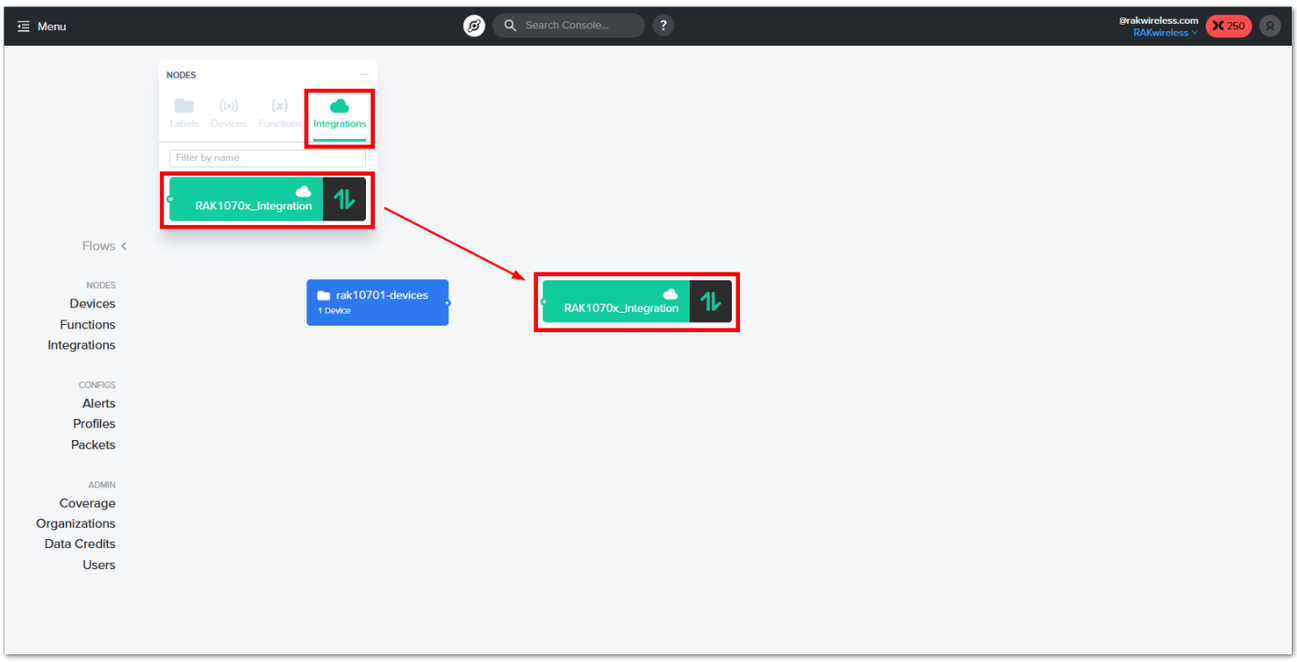

Figure 24: Drag the RAK1070x_Integration

13. You then have to connect the `Label` block to the `Integration` block via the tiny connector indicated by the red arrows by using your mouse cursor and dragging the line connector.

> 📝 **NOTE:**
>
> There is no need to save the changes created on the flows canvas since it is automatically saved as you do changes.

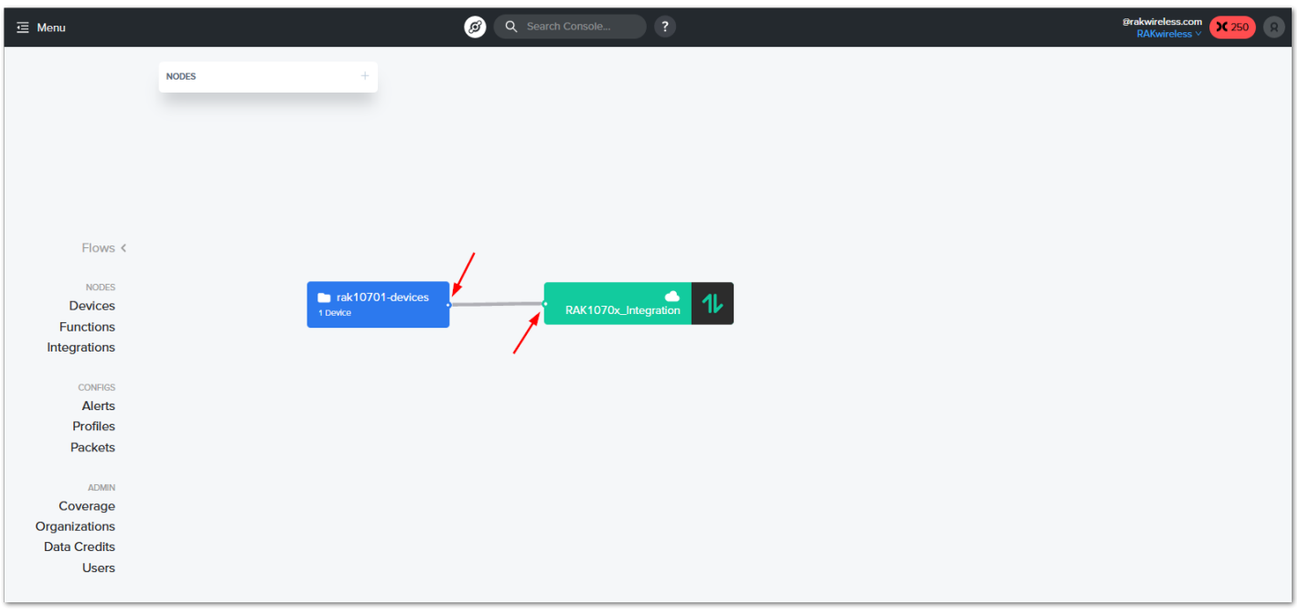

Figure 25: Connecting Labels to Integrations

14. The final step in the setting up of the Helium Console for RAK10701 is the setting up of packets. You have to click on `Packets` and then the `Add New Packet Config` icon.

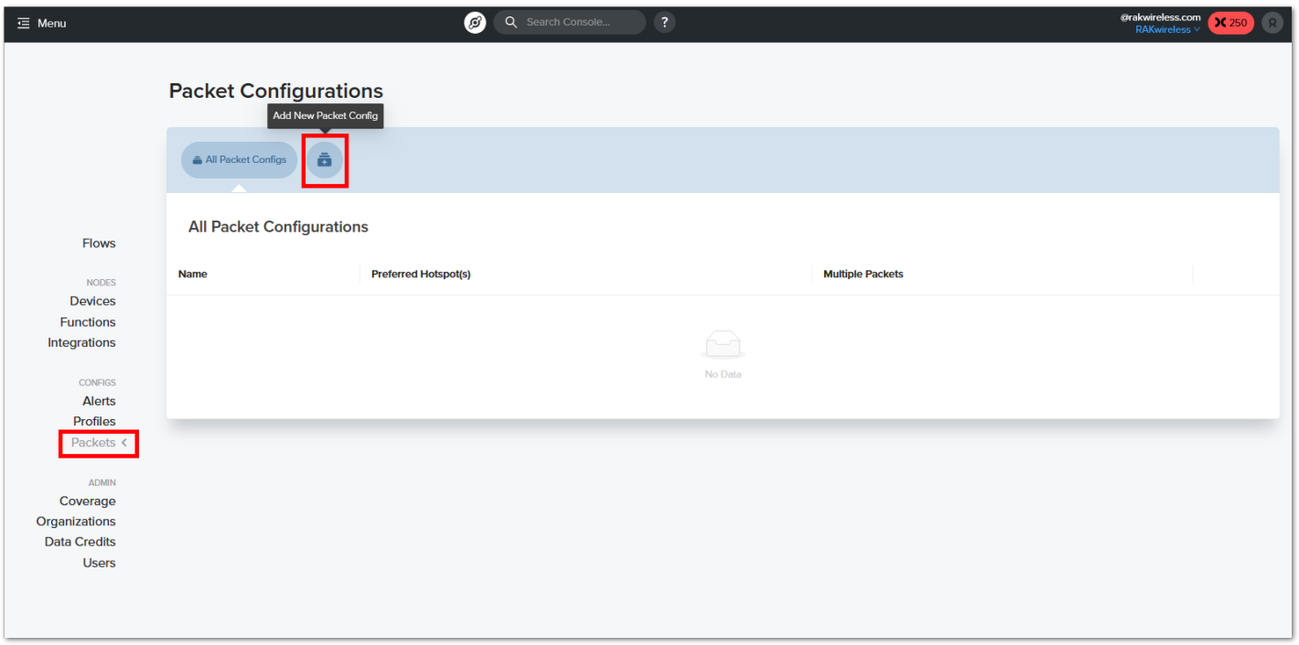

Figure 26: Creating Packets

15. You must also input a name on the `Packet Config Name`, select `Multiple Packets` and drag the slider so it will show `All Available Packets`. Once done, you can now click on `+ Create Packet Config`.

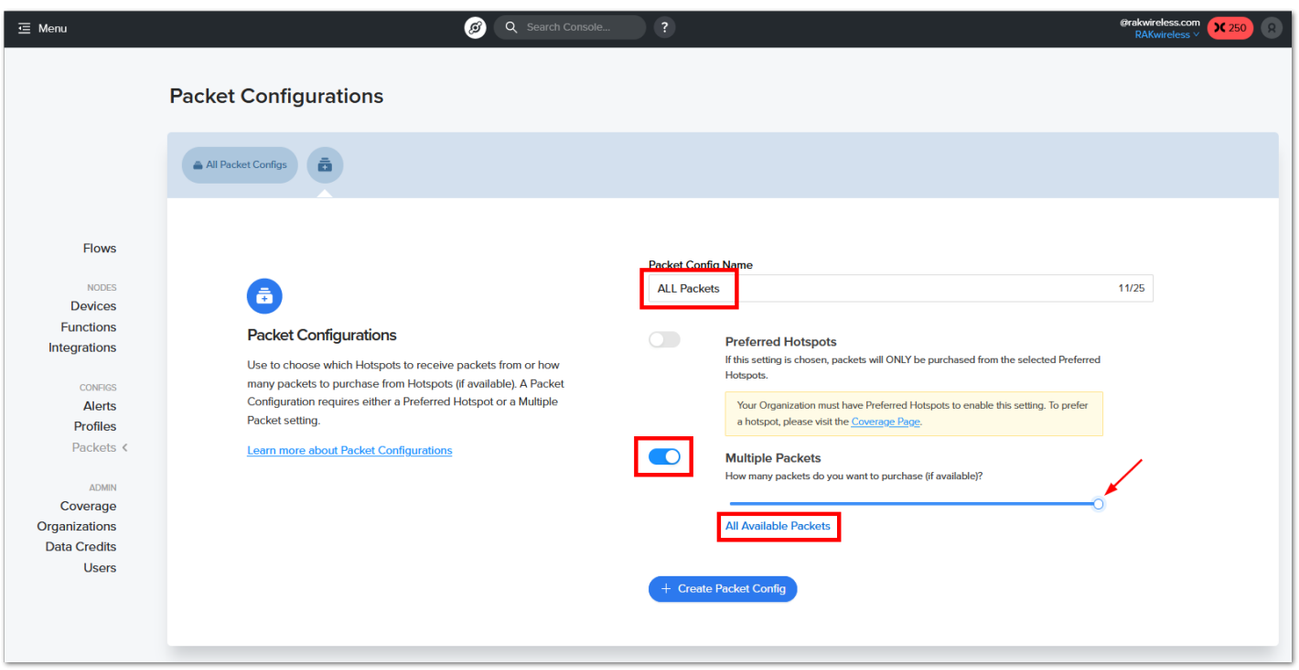

Figure 27: Multiple packets configuration

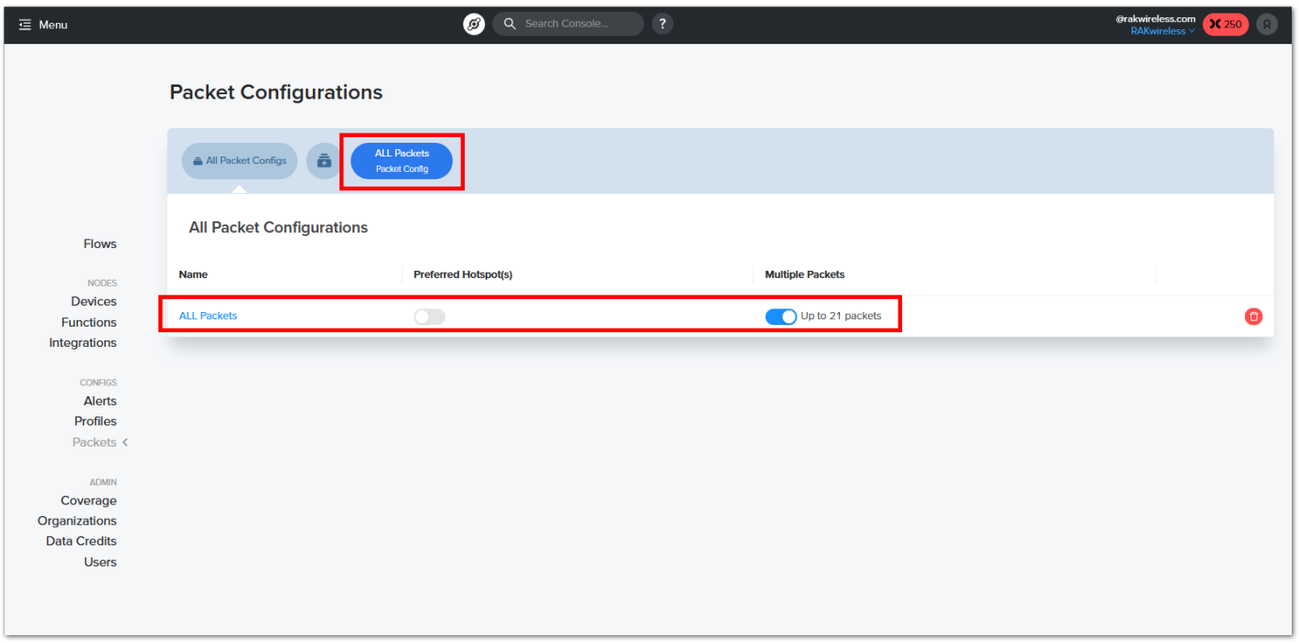

Figure 28: Successful packets creation

16. You must associate the multiple packet setup with the `rak10701-devices` label. To do this, you have to go back on `Flows`, double-click on the `rak10701-devices` label, choose the `Pakets` tab under rak10701-devices settings then enable `ALL Packets`. After this, you can now proceed with the configuration of RAK10701 using WisToolBox.

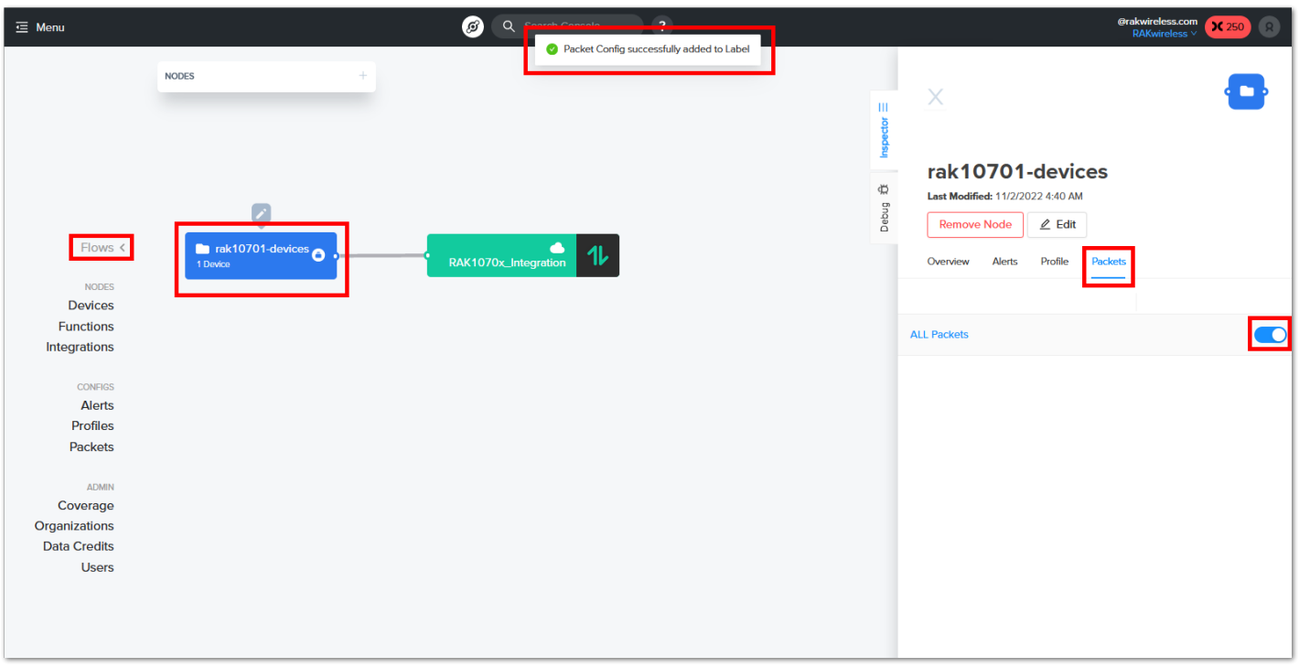

Figure 29: Attaching multiple packets to the rak10701-devices label

17. You can now proceed on device configuration so that the proper EUIs and KEY will match the one in the Helium network.

RAK10701-P Field Tester Pro Guide for The Things Network

This section shows how to use the RAK10701-P Field Tester Pro for LoRaWAN to The Things Stack.

1. Log in to TTNv3. To do so, head to the TTNv3 site and select your cluster. If you already have a TTN account, you can use your The Things ID credentials to log in.

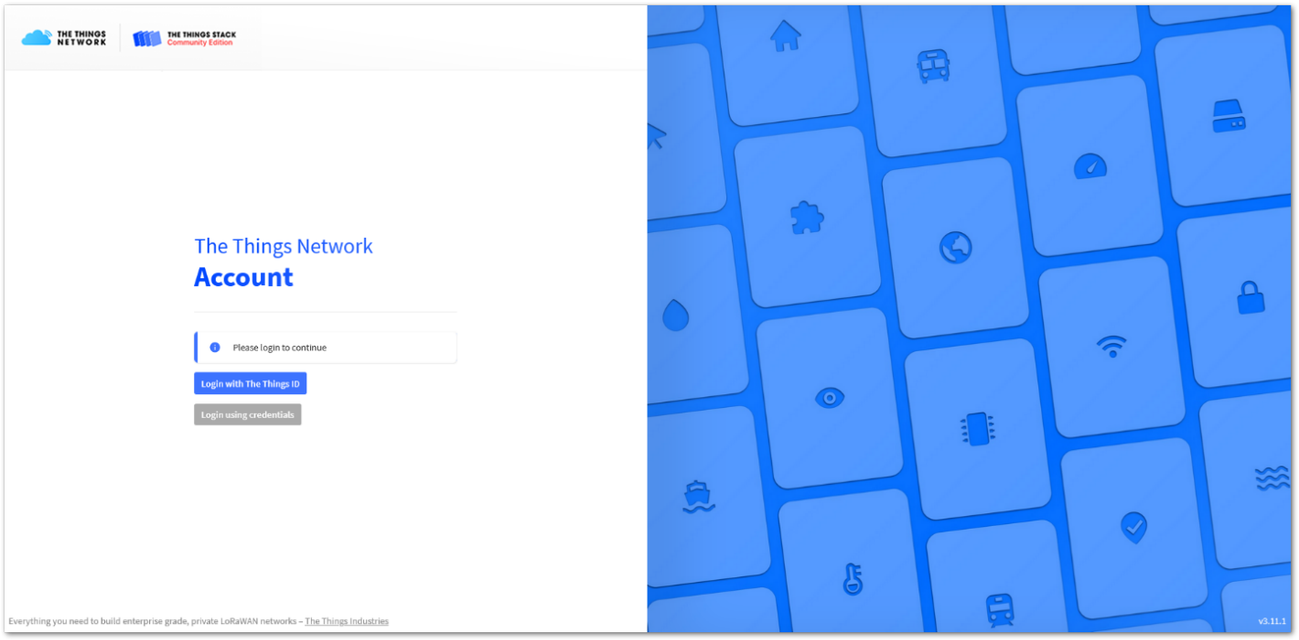

Figure 30: The Things Stack home page

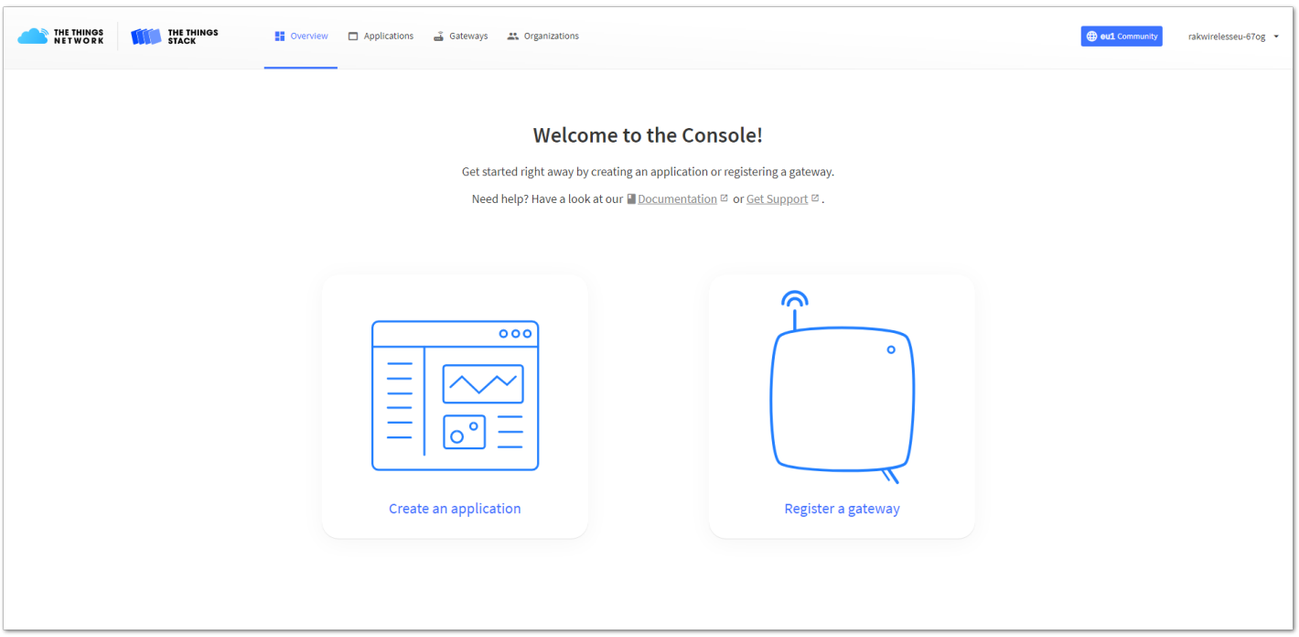

Figure 31: Console page after a successful login

> 📝 **NOTE:**
>
> To connect RAK10701-P Field Tester Pro to TTNv3, you should already have connected a gateway in range to TTNv3. Or, you have to be sure that you are in the range of a public gateway.

2. Now that you are logged in to the platform, the next step is to create an application. In your console, click **Create an application**.

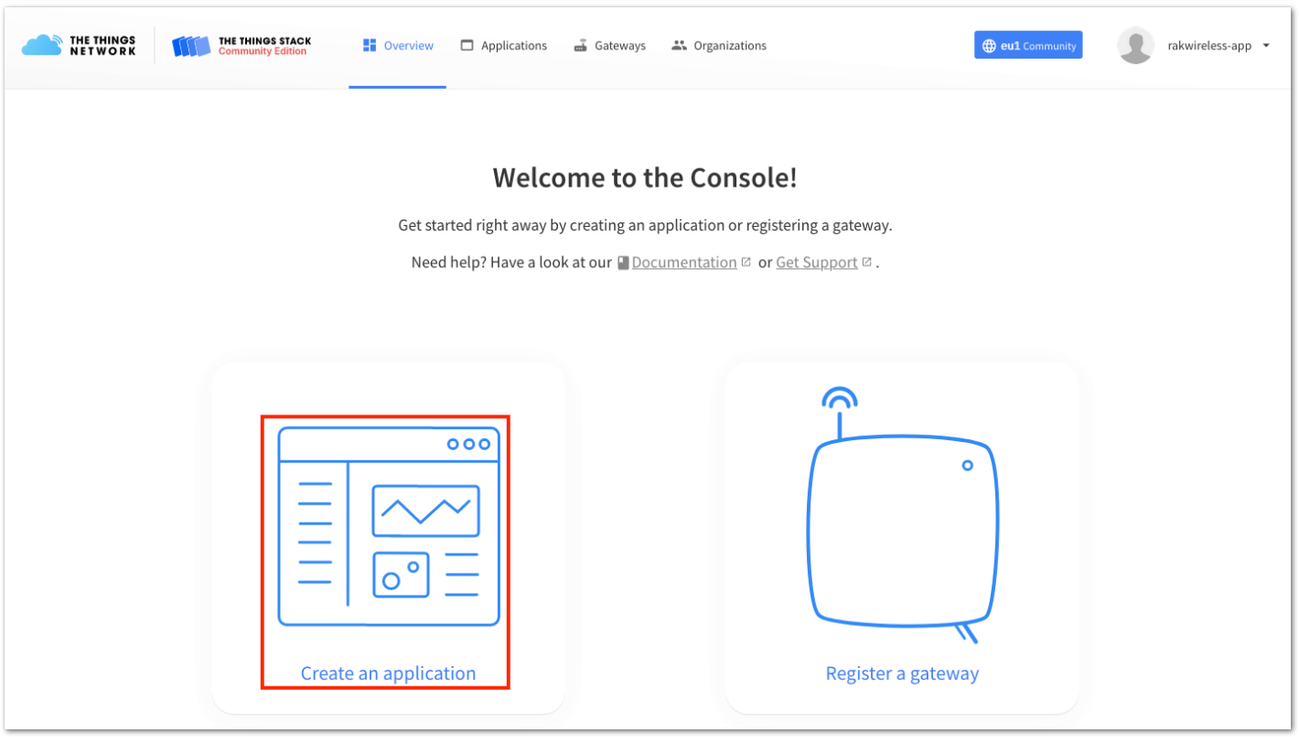

Figure 32: Create an application

3. To have an application registered, you need to input first the specific details and necessary information about your application then click **Create application**.

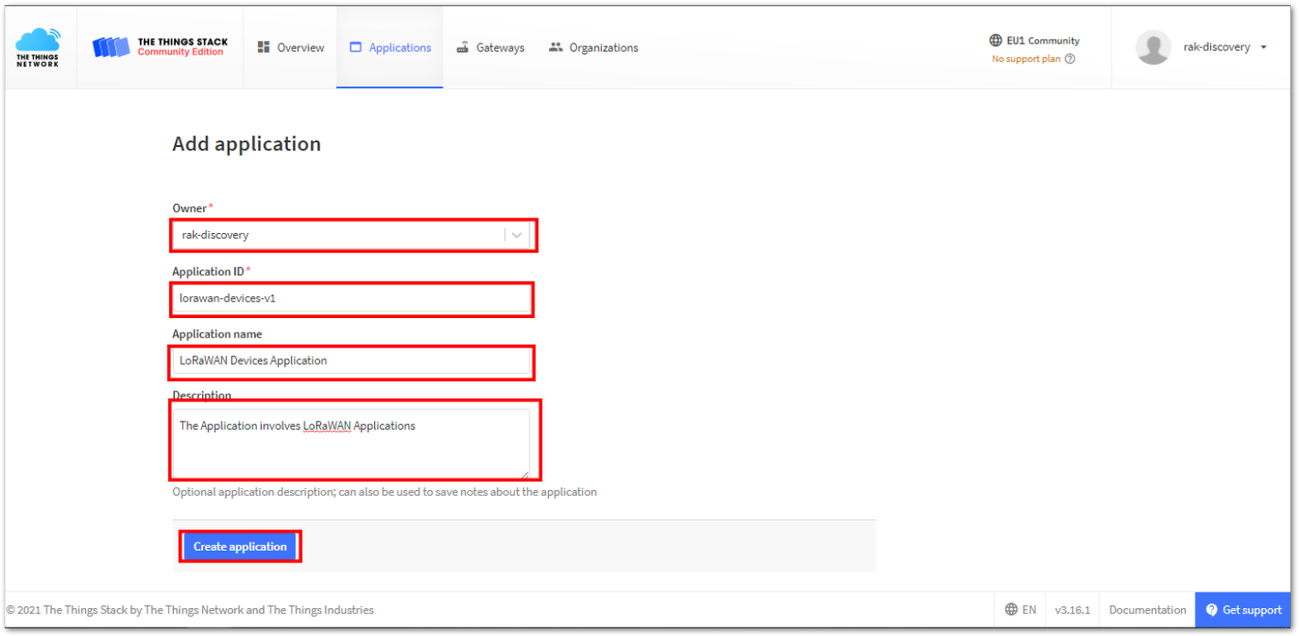

Figure 33: Creating an Application

4. If you had no error during the previous step, you should now be on the application console page. The next step is to **add end-devices to your TTN application**.

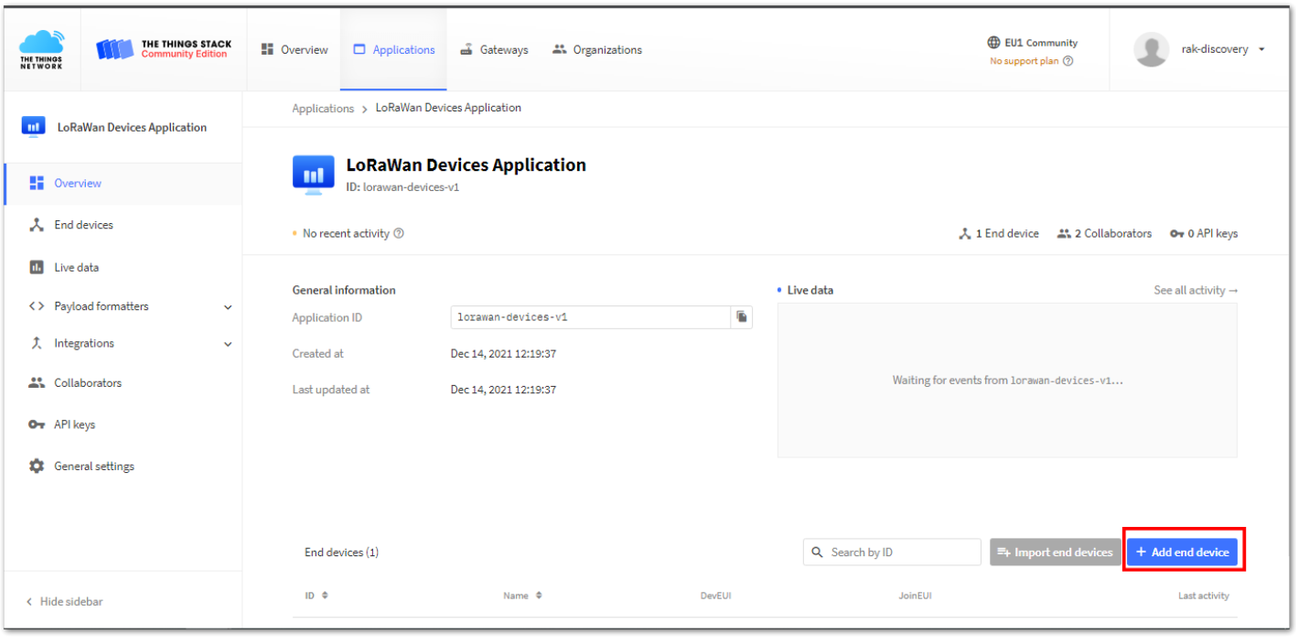

Figure 34: Add end-devices to your TTN application

5. To register the RAK10701-P Field Tester Pro, you need to click **Manually** first.

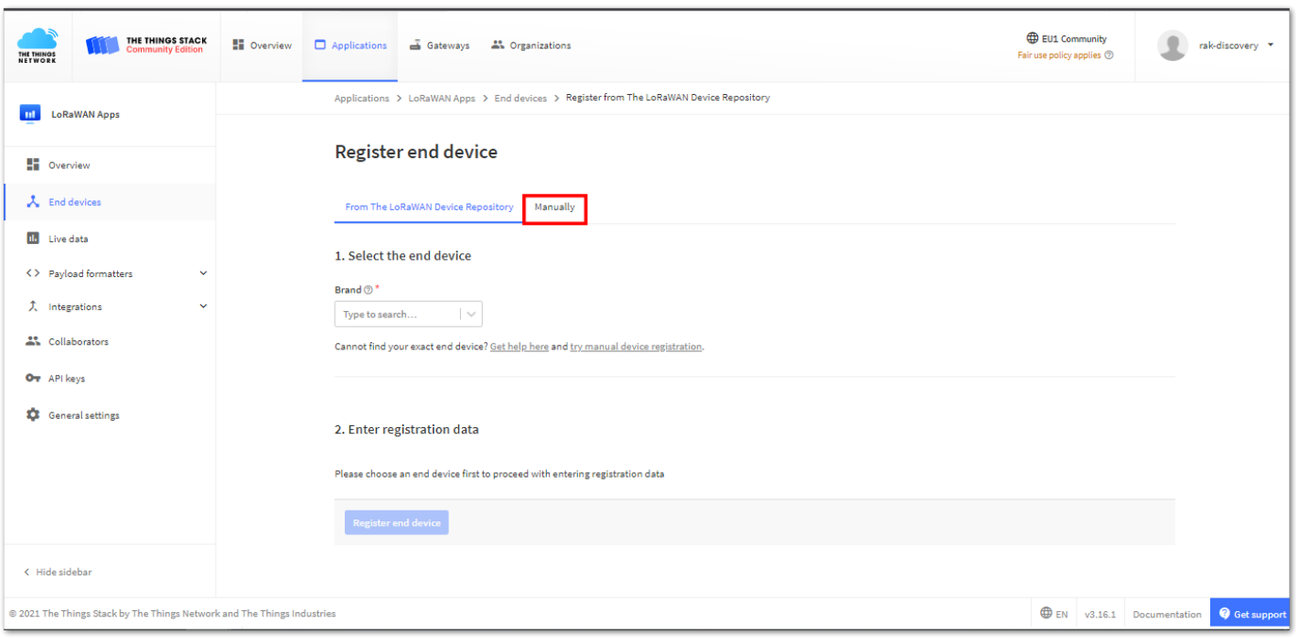

Figure 35: Adding end devices manually

6. Choose the following configurations in adding the end devices. You must choose the correct Frequency Plan and the LoRaWAN version must be 1.0.3.

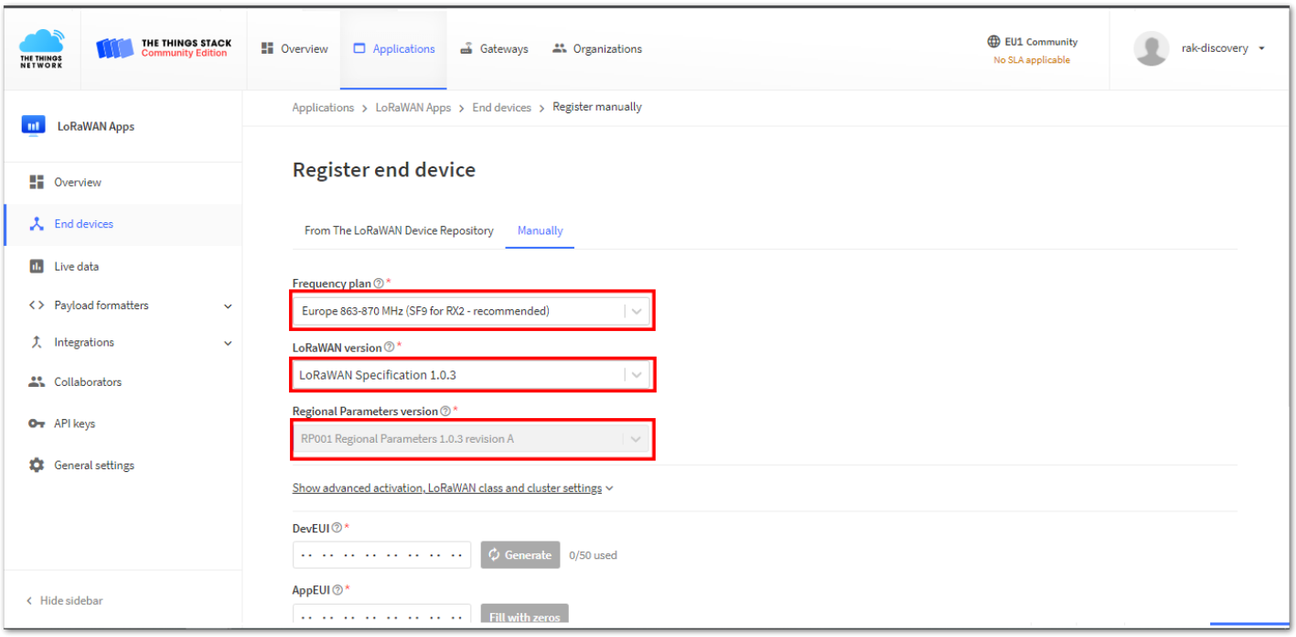

Figure 36: Configurations for adding end devices

7. Click **Show advanced activation, LoRaWAN class, and cluster settings**, then select **Over the air action** (OTAA).

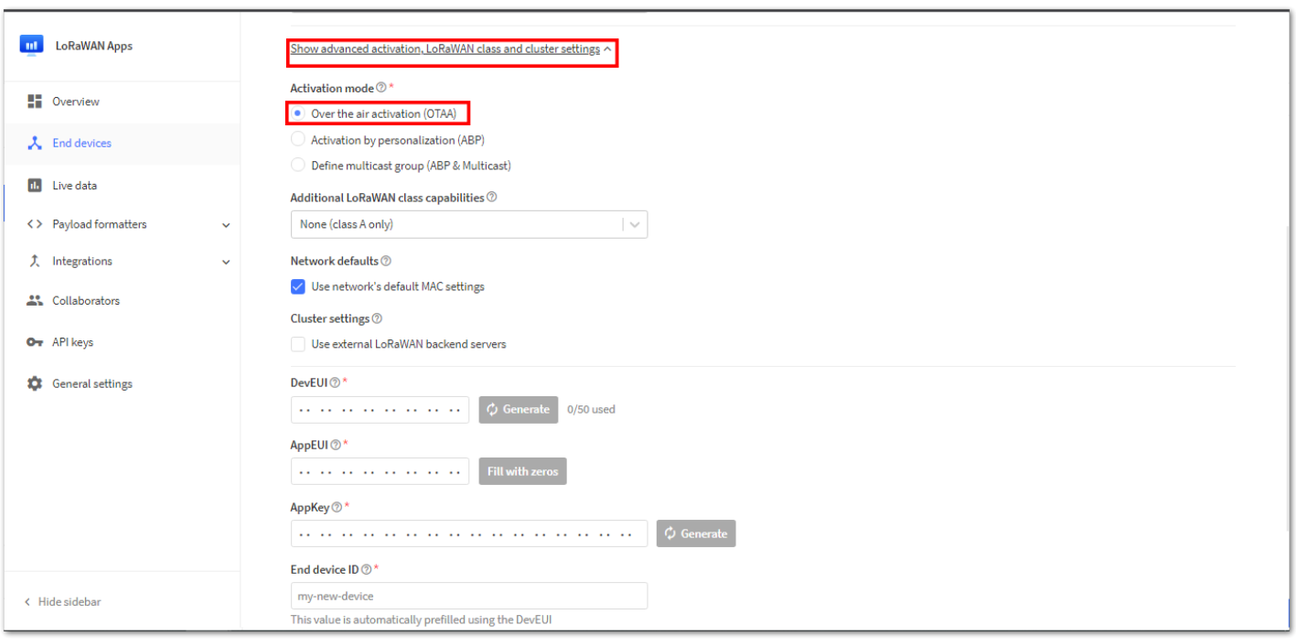

Figure 37: OTAA settings

8. Then input the LoRaWAN OTAA parameters. For **AppEUI**, you may click **Fill with Zeros**. For **AppKey** and **DevEUI**, you can click **Generate**. Then the parameters will be automatically filled by the TTS platform. Finally, click **Register End Device**.

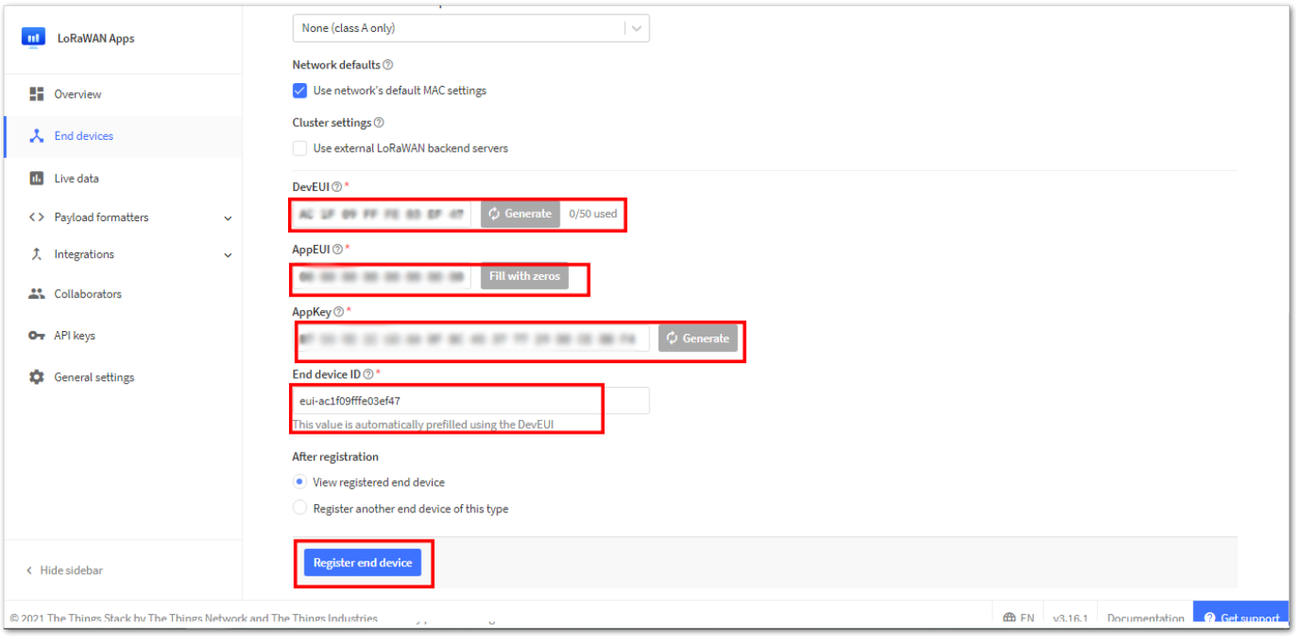

Figure 38: Registering the end device

9. You should now be able to see the device on the TTN console after you fully register your device. Take note of these OTAA parameters, such as the `AppEUI`, `DevEUI`, and the `AppKey`, as they are needed in the configuration of the RAK10701-P Field Tester Pro hardware later on in this guide.

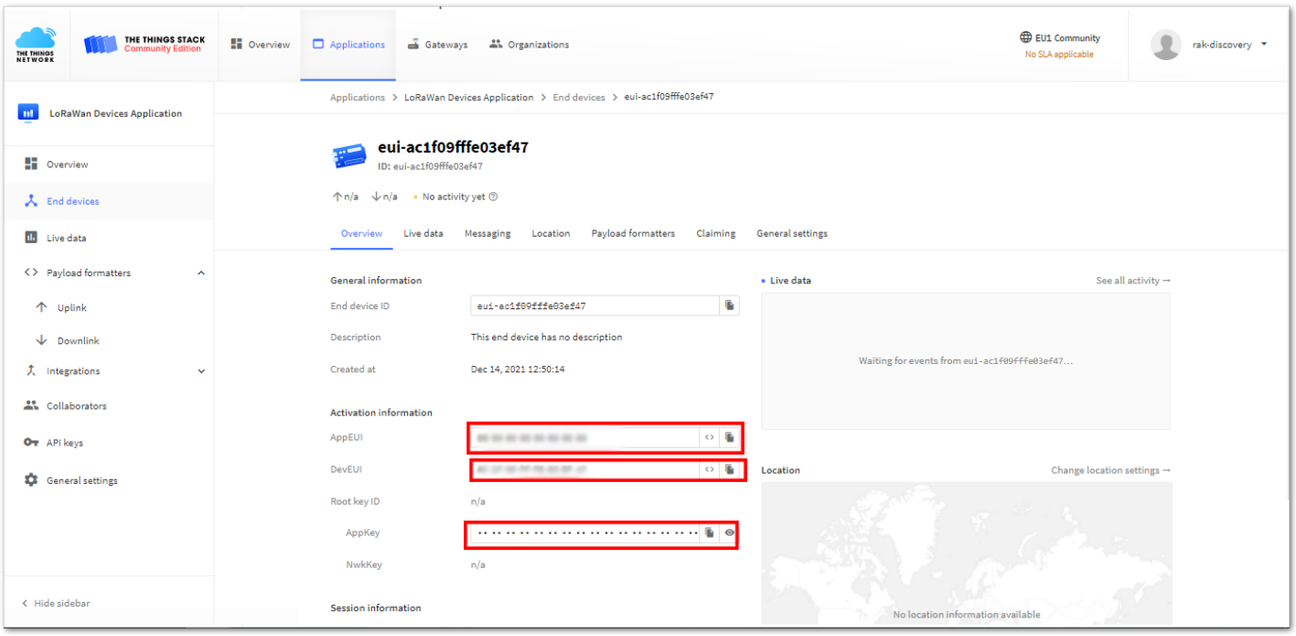

Figure 39: OTAA device successfully registered to TTN

10. After adding the device to the LoRaWAN application, link it to the backend server. The first step is to create an **API key**.

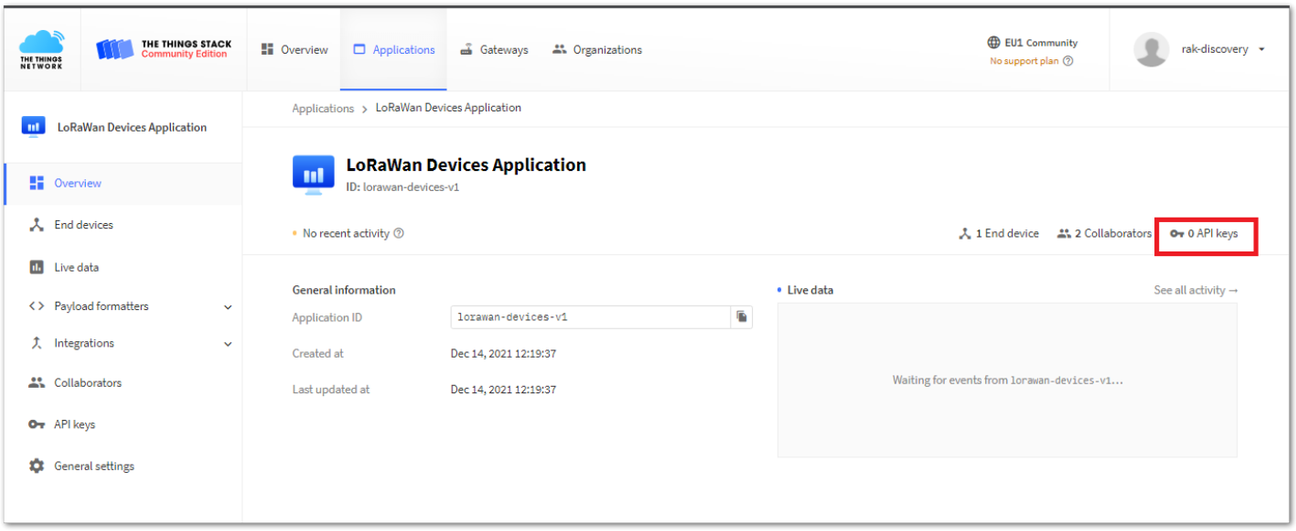

Figure 40: Creating API key

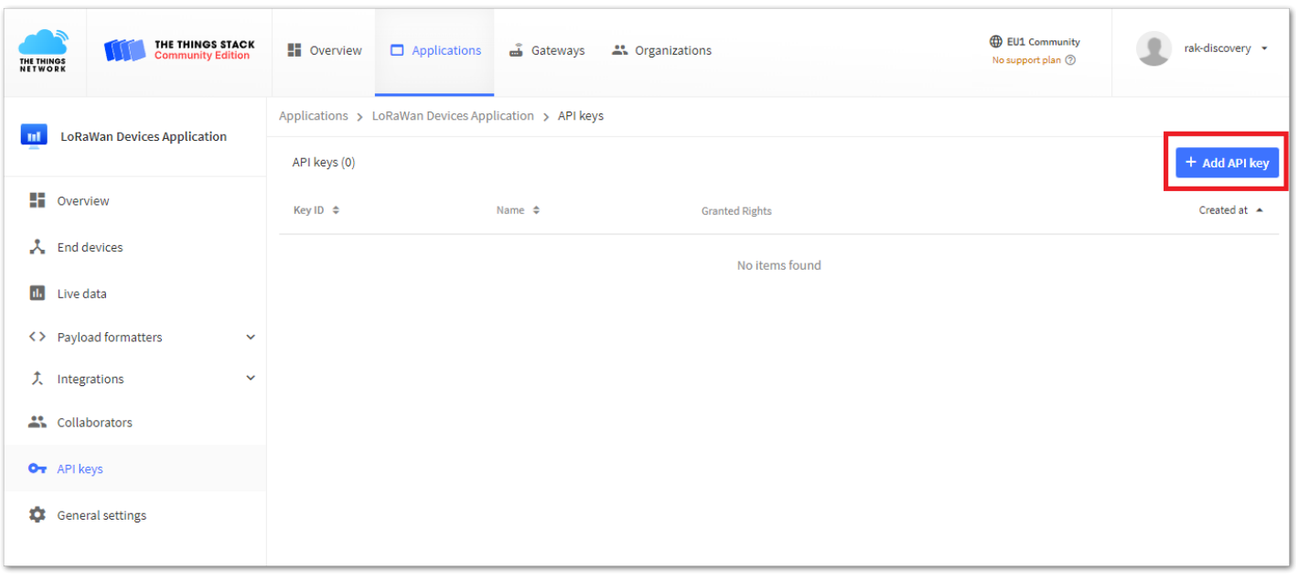

Figure 41: Creating API key

11. Configure the API key parameters. You can put any names that will easily track your API. You have to set the expiration date as well. Then you must check `Write downlink application traffic`. After the configuration, you can now click `Create API key`.

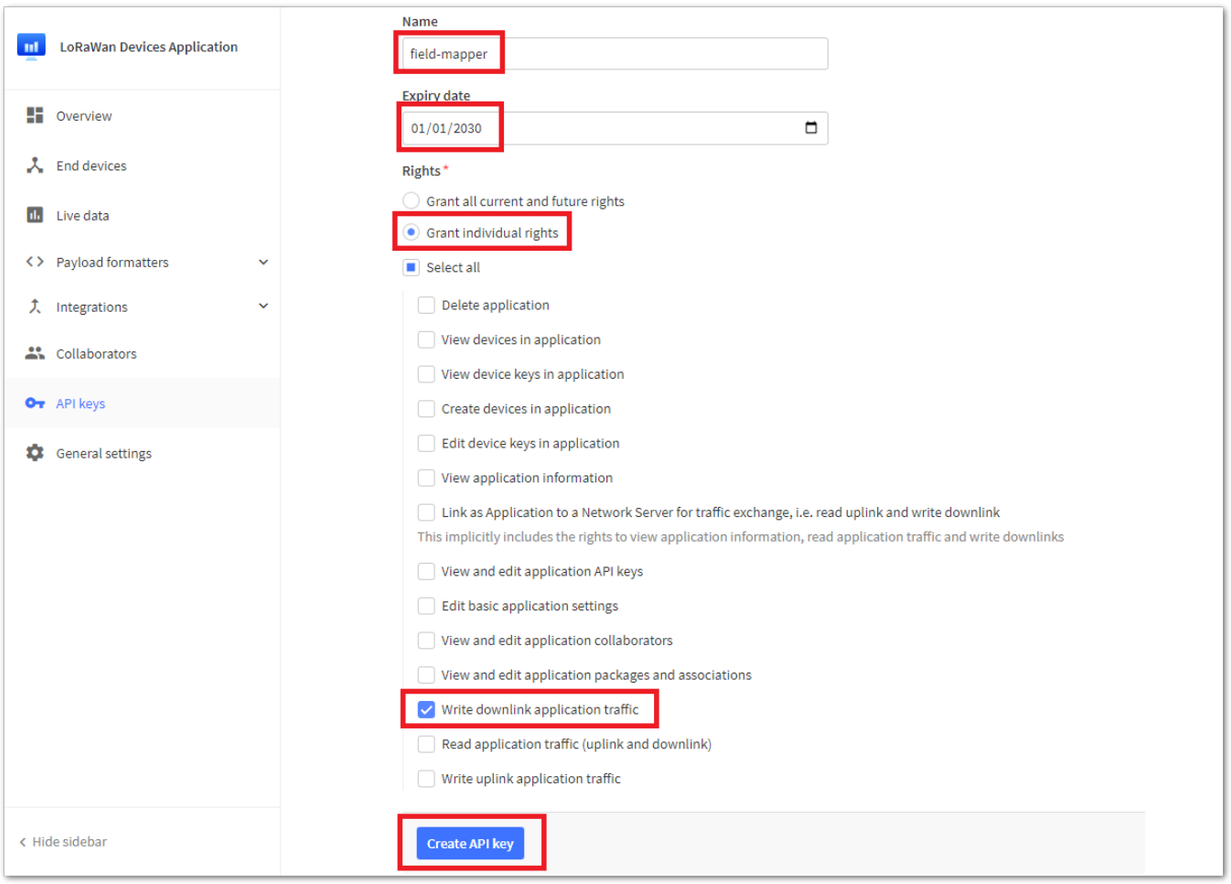

Figure 42: API key parameters

12. This step is critical. You need to copy the API key because this will be used on Webhook integration.

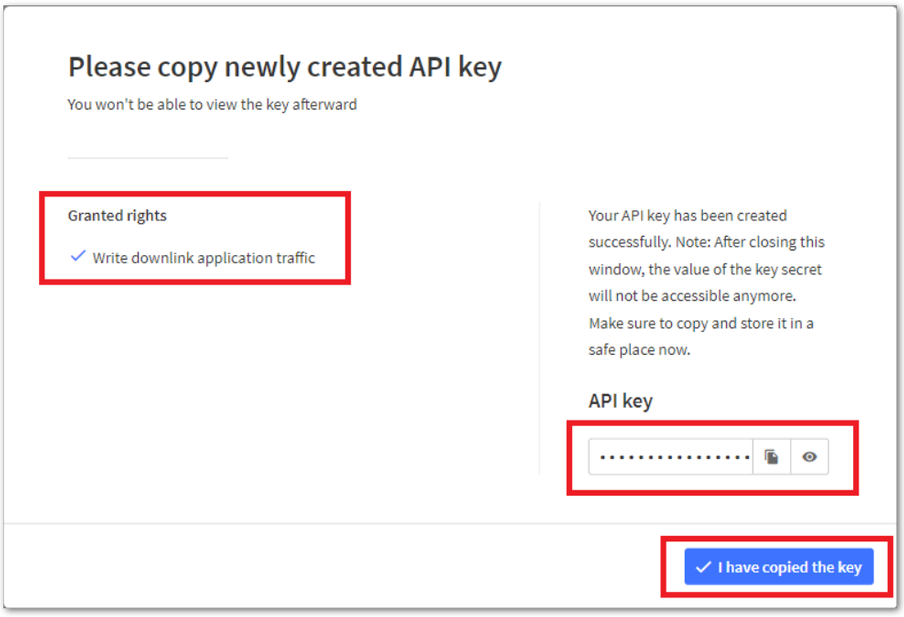

Figure 43: Copy API key

13. With the API key created, you can proceed with creating the Webhook integration.

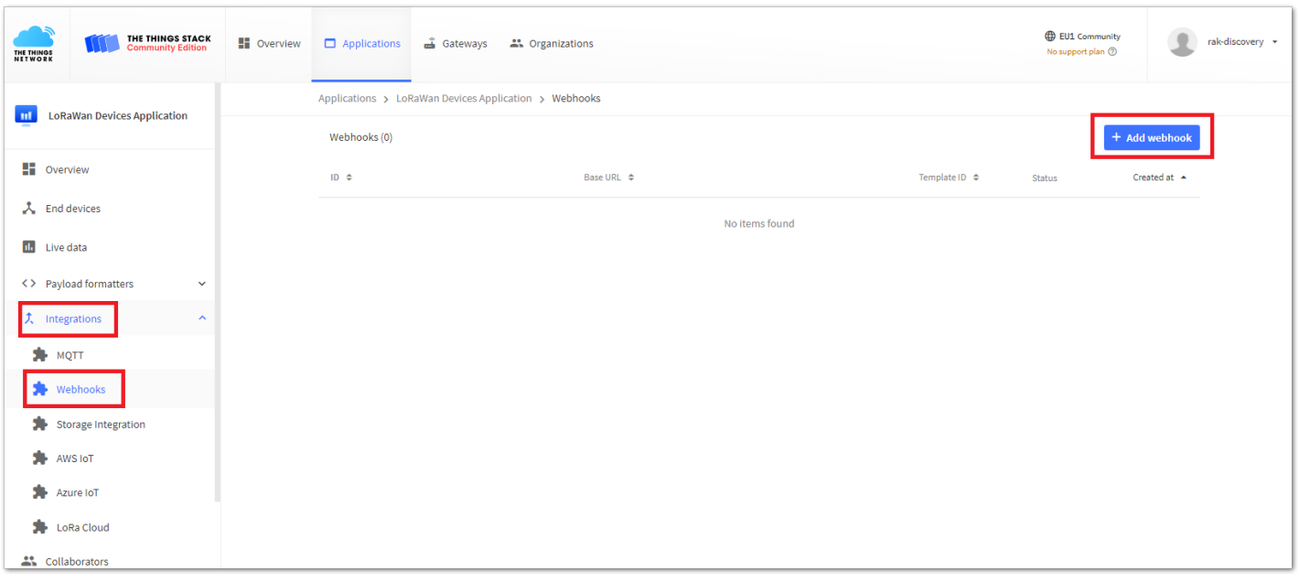

Figure 44: Webhook Integration

14. Select **Custom Webhook**.

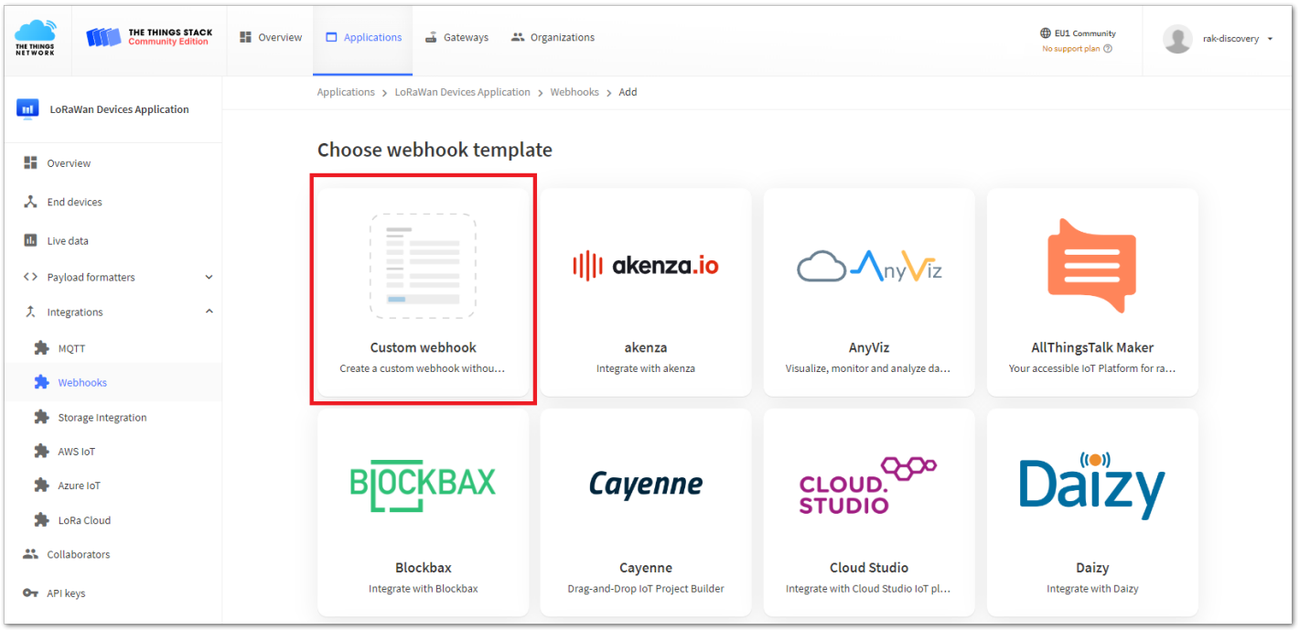

Figure 45: Custom Webhook

15. Configure the necessary parameters on the Webhook. You can select any name for the webhook. You then need to set the base URL going to disk19 server `https://dev.disk91.com/fieldtester/ttn/v3`, add the API key from the previous step and lastly put a check on the `Uplink message` under **Enabled event types**.

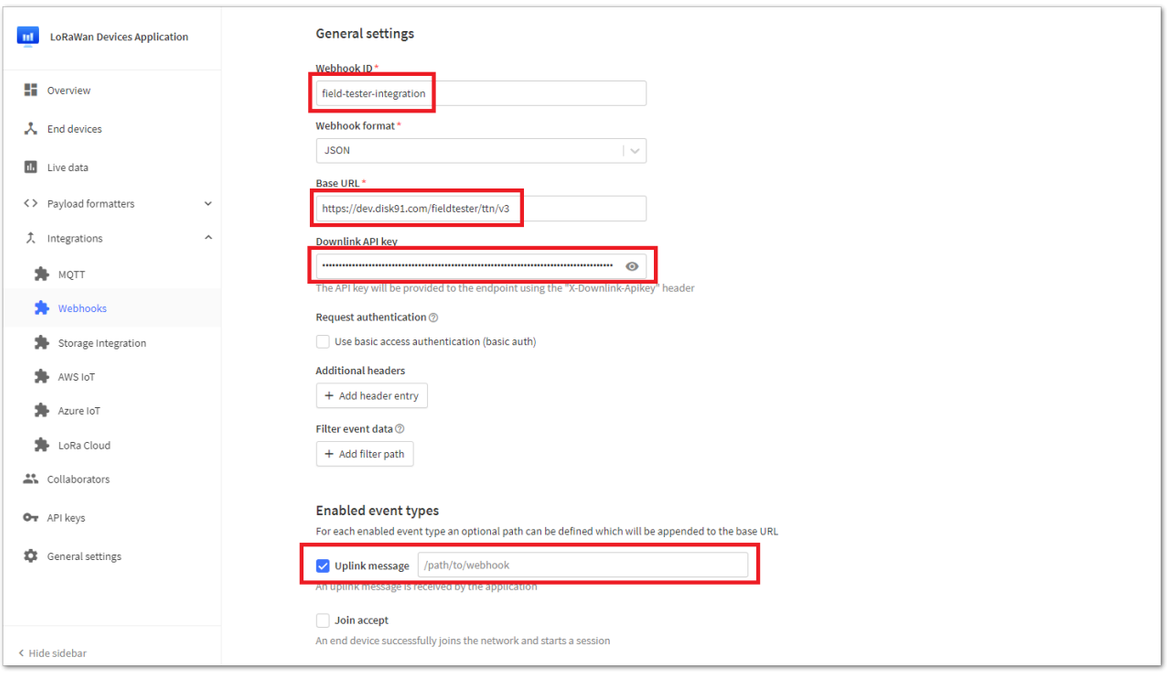

Figure 46: Webhook parameters

16. After setting all the configurations, you can now add the webhook.

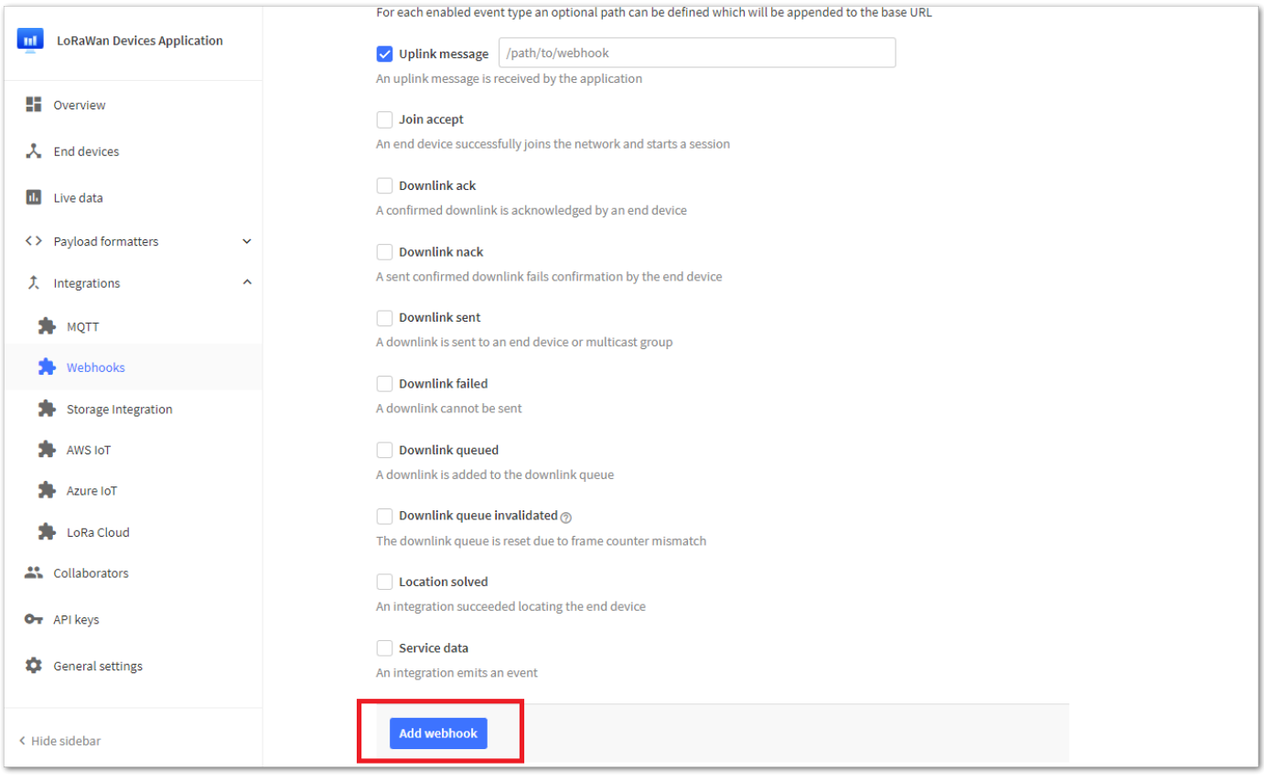

Figure 47: Add Webhook

17. You should see now the newly created webhook.

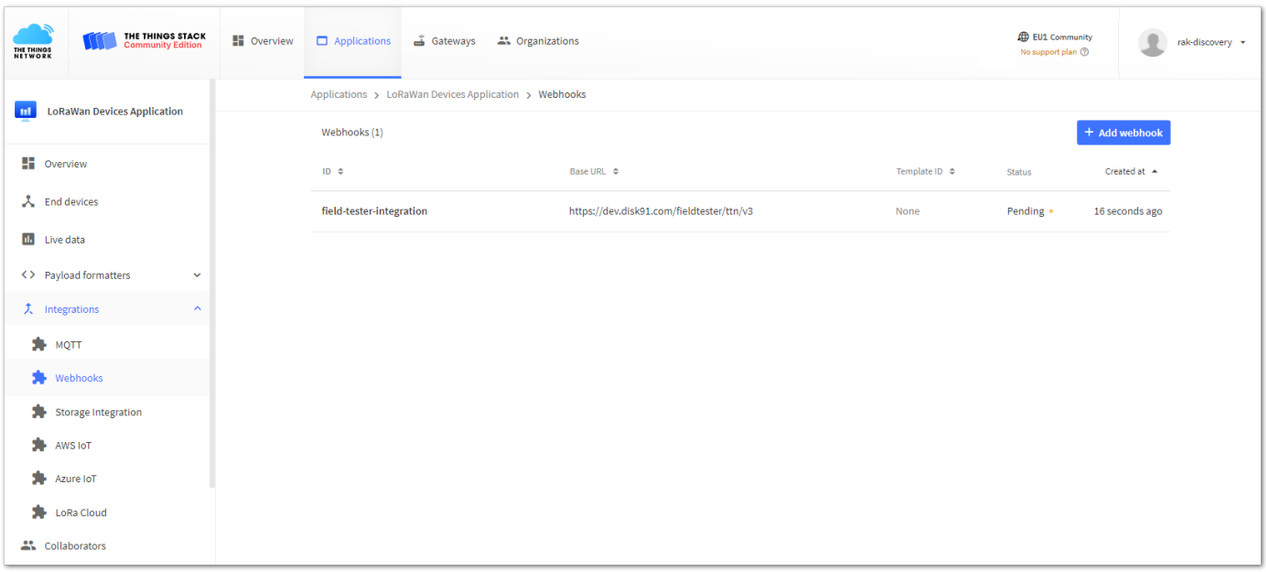

Figure 48: Add Webhook

18. After adding the application, device, and webhook integration to the console, you have to configure the parameters in your device to match the parameters on the TTN console. You can use WisToolBox via USB connection or wirelessly via BLE. You can now proceed on the RAK10701-P Configuration using WisToolBox. You also have the option to update device parameters directly via RUI3 AT Commands (if you prefer AT commands instead of WisToolBox).

19. Once you configured the RAK10701-P with the correct Frequency Band and EUIs/Key by following the guide on the RAK10701-P Configuration using WisToolBox, you should see the join request/accept, uplinks and downlinks to The Things Stack console. These uplinks contains the coordinates of the field tester and the downlinks contains the data calculated by the disk91 server. The uplink uses fport 1 and the downlink uses fport 2. To view the actual coordinates on the console, you need to add a payload decoder on your uplink data.

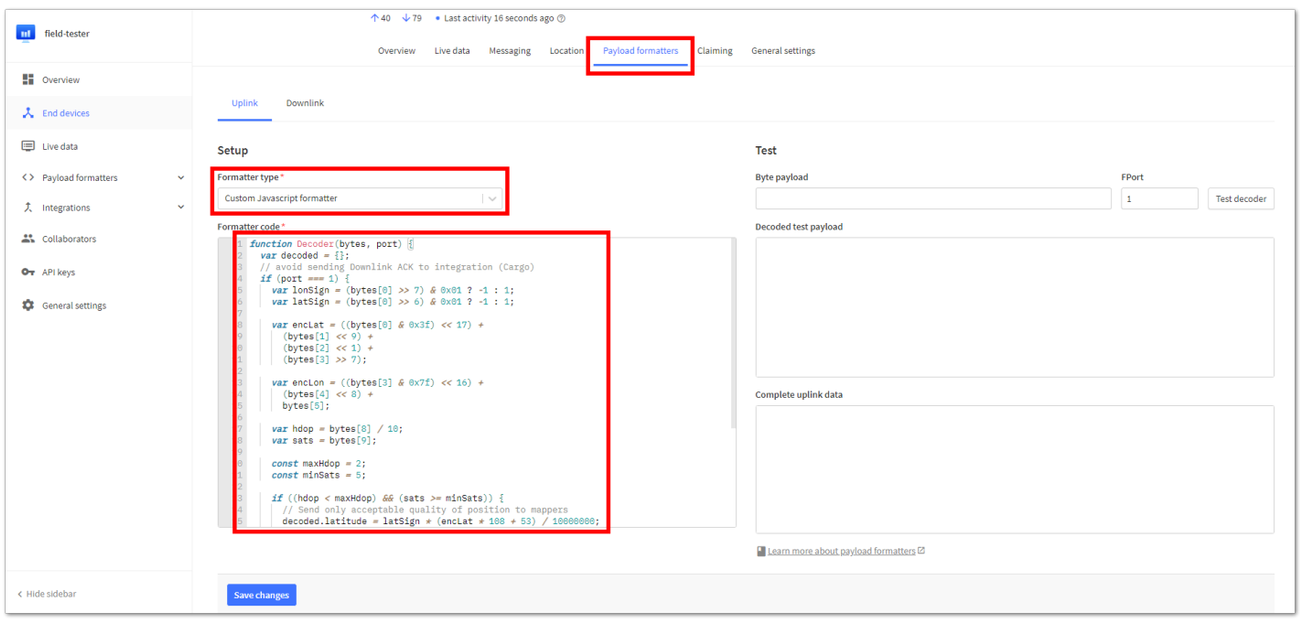

Figure 49: Adding Uplink Payload Decoder

Uplink payload decoder script.

```js
function Decoder(bytes, port) {
    var decoded = {};
    // avoid sending Downlink ACK to integration (Cargo)
    if (port === 1) {
        var lonSign = (bytes[0] >> 7) & 0x01 ? -1 : 1;
        var latSign = (bytes[0] >> 6) & 0x01 ? -1 : 1;

        var encLat = ((bytes[0] & 0x3f) << 17) +
            (bytes[1] << 9) +
            (bytes[2] << 1) +
            (bytes[3] >> 7);

        var encLon = ((bytes[3] & 0x7f) << 16) +
            (bytes[4] << 8) +
            bytes[5];

        var hdop = bytes[8] / 10;
        var sats = bytes[9];

        const maxHdop = 2;
        const minSats = 5;

        if ((hdop < maxHdop) && (sats >= minSats)) {
            // Send only acceptable quality of position to mappers
            decoded.latitude = latSign * (encLat * 108 + 53) / 10000000;
            decoded.longitude = lonSign * (encLon * 215 + 107) / 10000000;
            decoded.altitude = ((bytes[6] << 8) + bytes[7]) - 1000;
            decoded.accuracy = (hdop * 5 + 5) / 10
            decoded.hdop = hdop;
            decoded.sats = sats;
        } else {
            decoded.error = "Need more GPS precision (hdop must be <" + maxHdop +
                " & sats must be >= " + minSats + ") current hdop: " + hdop + " & sats:" + sats;
        }
        return decoded;
    }
    return null;
}
```

20. With the correct payload decoder, you should now see GPS coordinates data which you can also use to other integration.

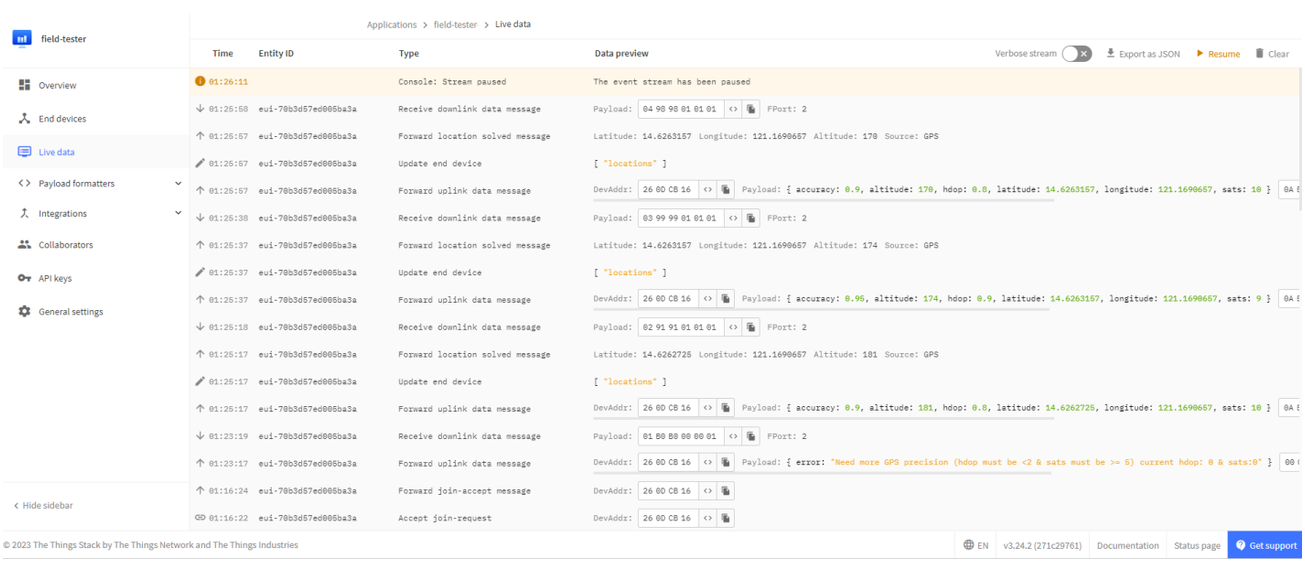

Figure 50: Decoded payload and TTS console

RAK10701-P Field Tester Pro Guide for Chirpstack

How Does It Work?

There are two steps under the hood of the Field Tester. In step one, the Field Tester is sending out data packets over LoRaWAN. These packets are received by one or multiple gateways. These packets are forwarded from the LoRaWAN network server to another backend server. When the packets are forwarded, they include information about signal strength and the number of gateways that have received the packet.

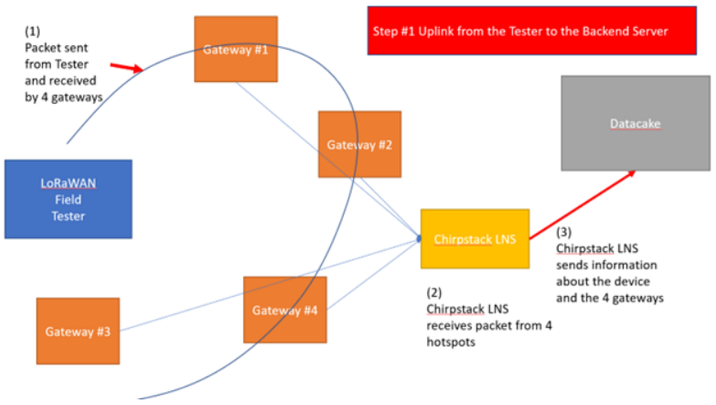

Figure 51: Step 1 - Field Tester Sending Uplink Payload

In the second step, the backend server is calculating the minimum and maximum distance between the Field Tester Pro and the gateways that received the data. Together with the minimum and maximum RSSI levels, this information is then sent back to the Field Tester Pro as a LoRaWAN downlink.

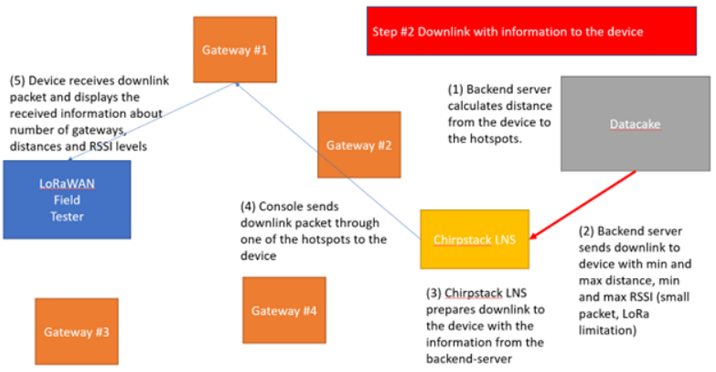

Figure 52: Step 2 - Backend Server Sending Useful Information as Downlink

To use Chirpstack for RAK10701-P, you must have a working installation of the Chirpstack LoRaWAN network server. It can be on a dedicated machine, Raspberry Pi, or in a cloud VPS instance. It should have a fixed IP address and port to where `Datacake.co` will connect to.

1. To start with Chirpstack, you must create a device profile for your RAK10701-P Field Tester Pro device. You must select `LoRaWAN MAC version 1.0.3` which is the LoRaWAN specification version that the RAK10701 Field Tester supports.

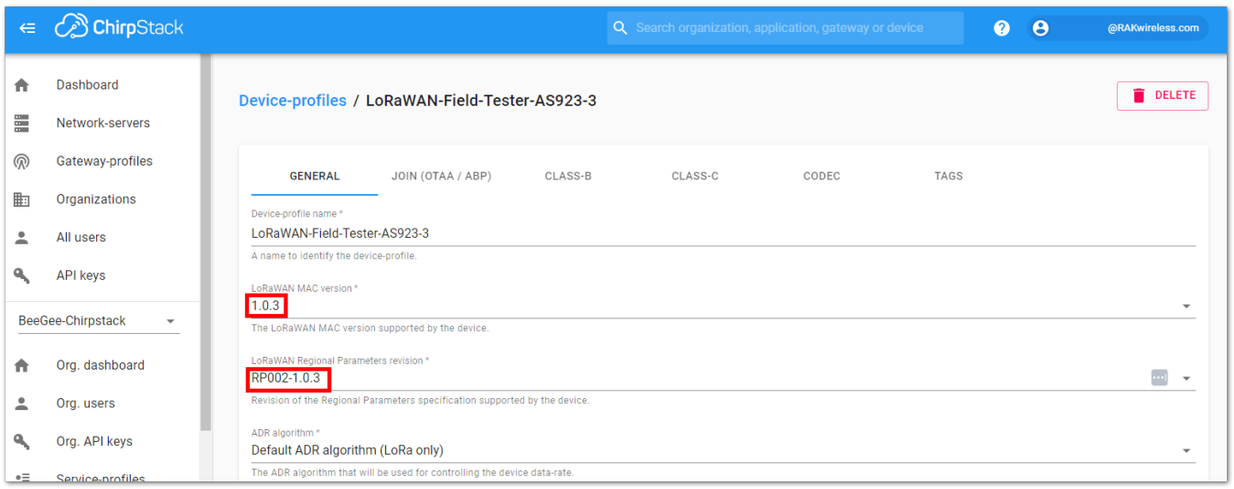

Figure 53: Creating Device Profile in Chirpstack

2. You must enable `Device supports OTAA` as the network join method as well.

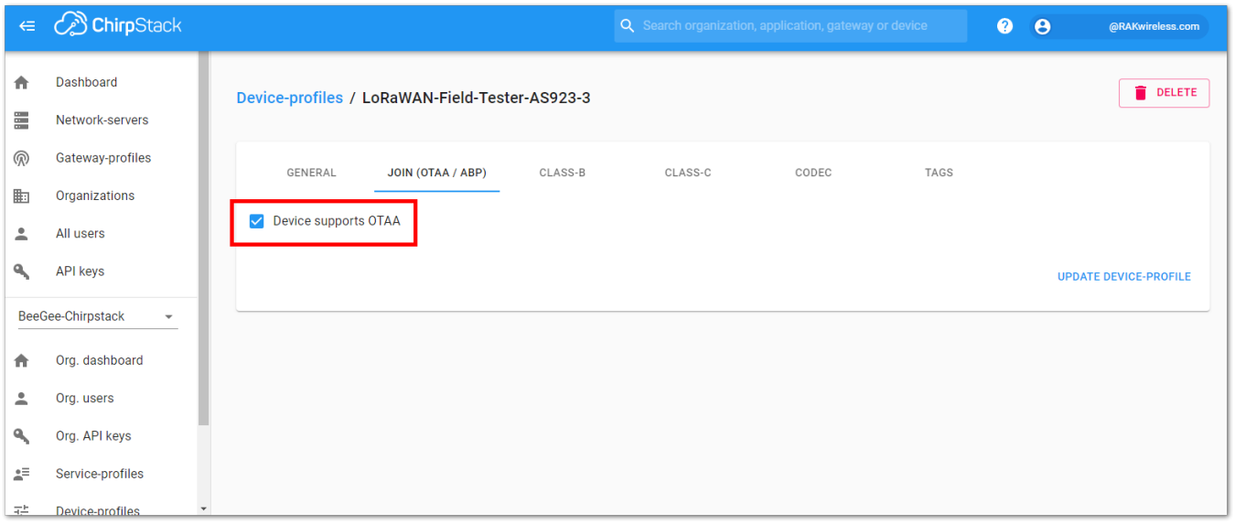

Figure 54: Enable support for OTAA

3. You can also include a custom javascript decoder under the `CODEC` tab. This will allow you to see the specific information transmitted by the device.

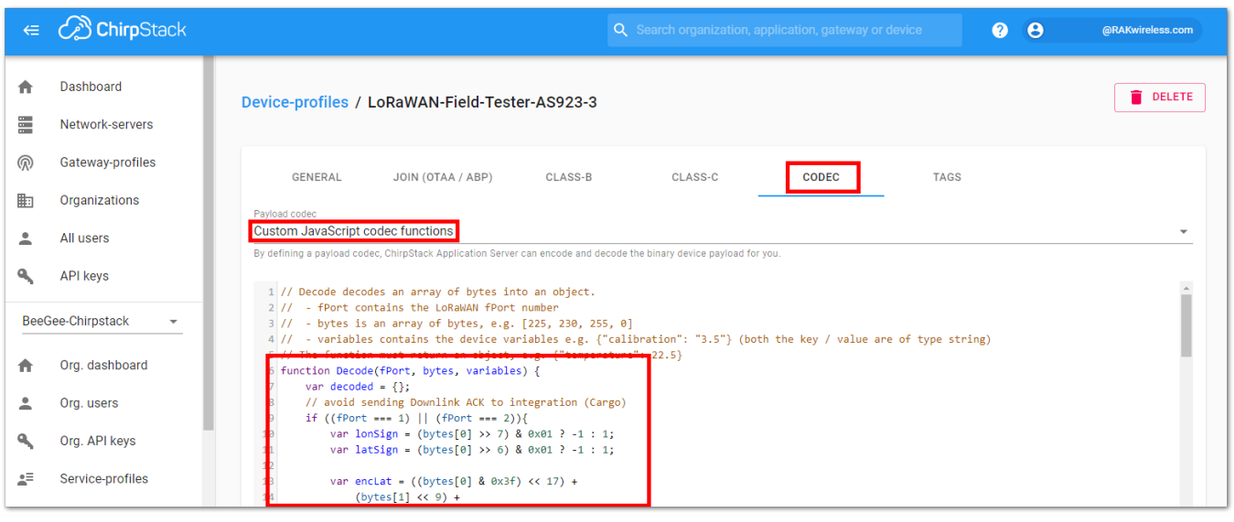

Figure 55: Custom Javascript Decoder for RAK10701 Field Mapper

Here's the complete decoder script:

```js
function Decode(fPort, bytes, variables) {
    var decoded = {};
    // avoid sending Downlink ACK to integration (Cargo)
    if ((fPort === 1) || (fPort === 2)){
        var lonSign = (bytes[0] >> 7) & 0x01 ? -1 : 1;
        var latSign = (bytes[0] >> 6) & 0x01 ? -1 : 1;

        var encLat = ((bytes[0] & 0x3f) << 17) +
            (bytes[1] << 9) +
            (bytes[2] << 1) +
            (bytes[3] >> 7);

        var encLon = ((bytes[3] & 0x7f) << 16) +
            (bytes[4] << 8) +
            bytes[5];

        var hdop = bytes[8] / 10;
        var sats = bytes[9];

        var maxHdop = 2;
        var minSats = 5;

        if ((hdop < maxHdop) && (sats >= minSats)) {
            // Send only acceptable quality of position to mappers
            decoded.latitude = latSign * (encLat * 108 + 53) / 10000000;
            decoded.longitude = lonSign * (encLon * 215 + 107) / 10000000;
            decoded.altitude = ((bytes[6] << 8) + bytes[7]) - 1000;
            decoded.accuracy = (hdop * 5 + 5) / 10
            decoded.hdop = hdop;
            decoded.sats = sats;
        } else {
            decoded.error = "Need more GPS precision (hdop must be <" + maxHdop +
                " & sats must be >= " + minSats + ") current hdop: " + hdop + " & sats:" + sats;
            decoded.latitude = latSign * (encLat * 108 + 53) / 10000000;
            decoded.longitude = lonSign * (encLon * 215 + 107) / 10000000;
            decoded.altitude = ((bytes[6] << 8) + bytes[7]) - 1000;
            decoded.accuracy = (hdop * 5 + 5) / 10
            decoded.hdop = hdop;
            decoded.sats = sats;
        }
        return decoded;
    }
    return null;
```

> 📝 **NOTE:**
>
> This decoder script can be found on RAKwireless Standardize Payload repository which also includes a custom decoder script for TTN and Helium.

4. After creating the device profile, you can now create an application and add the RAK10701 device. And then attached the `Device-profile` you created. You have to take note of the DEVEUI and APPKEY in this section. These parameters must match the ones in our RAK10701 Field Tester.

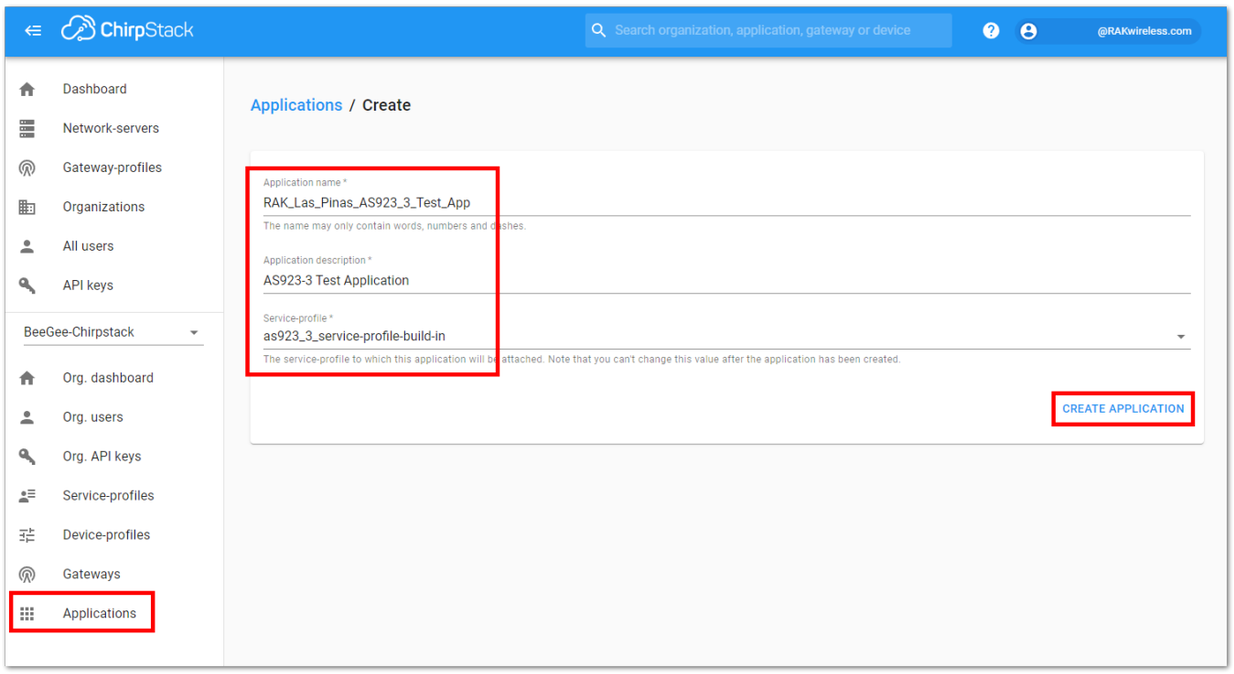

Figure 56: Create application in Chirpstack

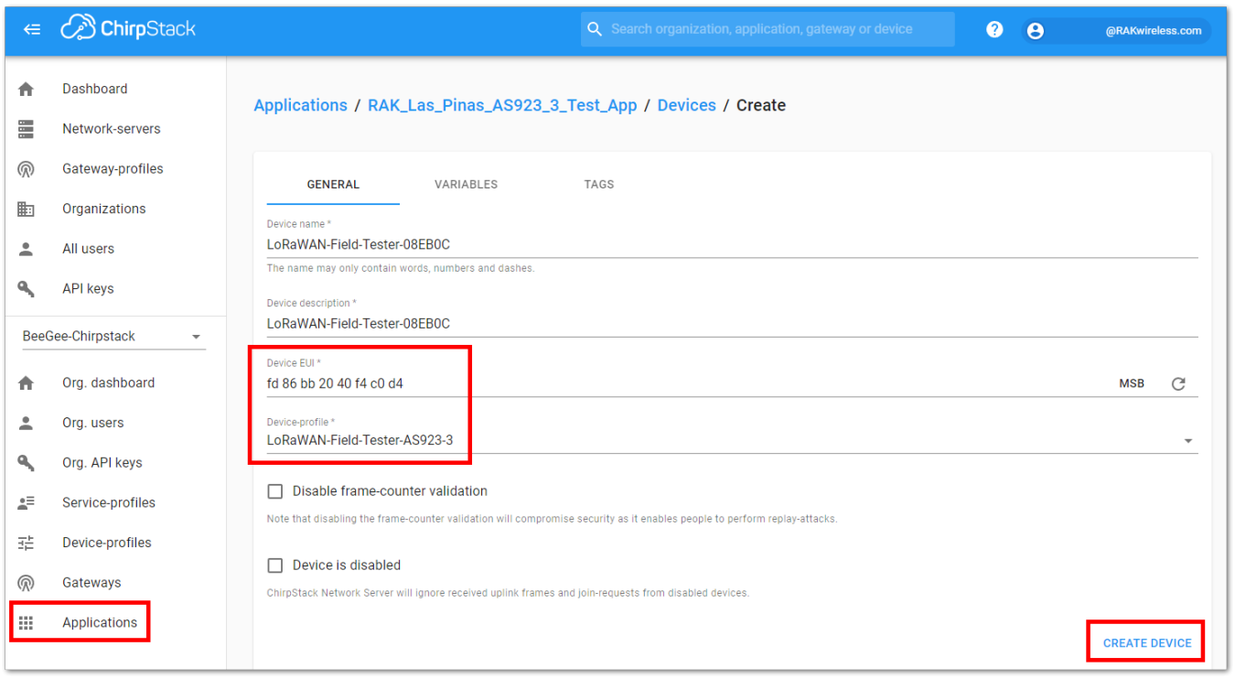

Figure 57: Create device in Chirpstack.

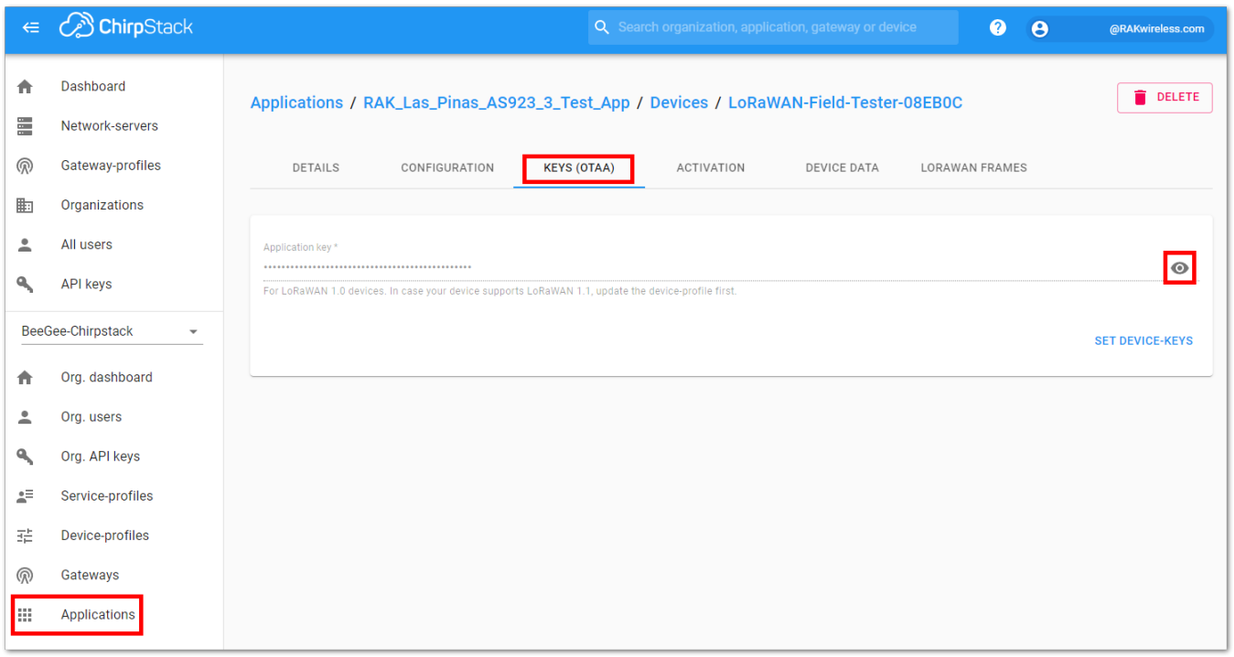

Figure 58: Device APPKEY

5. You also need to secure that you have a Gateway registered in Chirpstack and with the correct Network Server profile.

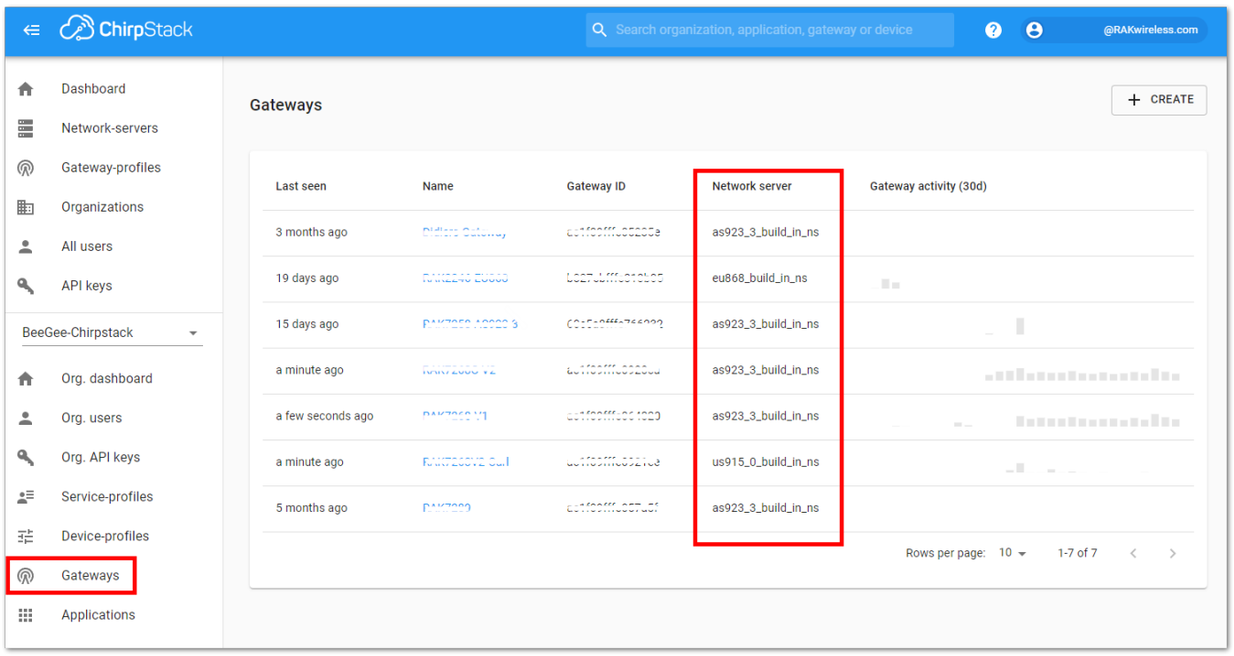

Figure 59: Gateways registered in Chripstack

6. The next step after setting up the network server, devices, and gateway, is the integration of the Chirpstack application to Datacake. You must choose HTTP, then click `Edit`. Then you have to use this endpoint going to datacake `https://api.datacake.co/integrations/lorawan/chirpstack/`.

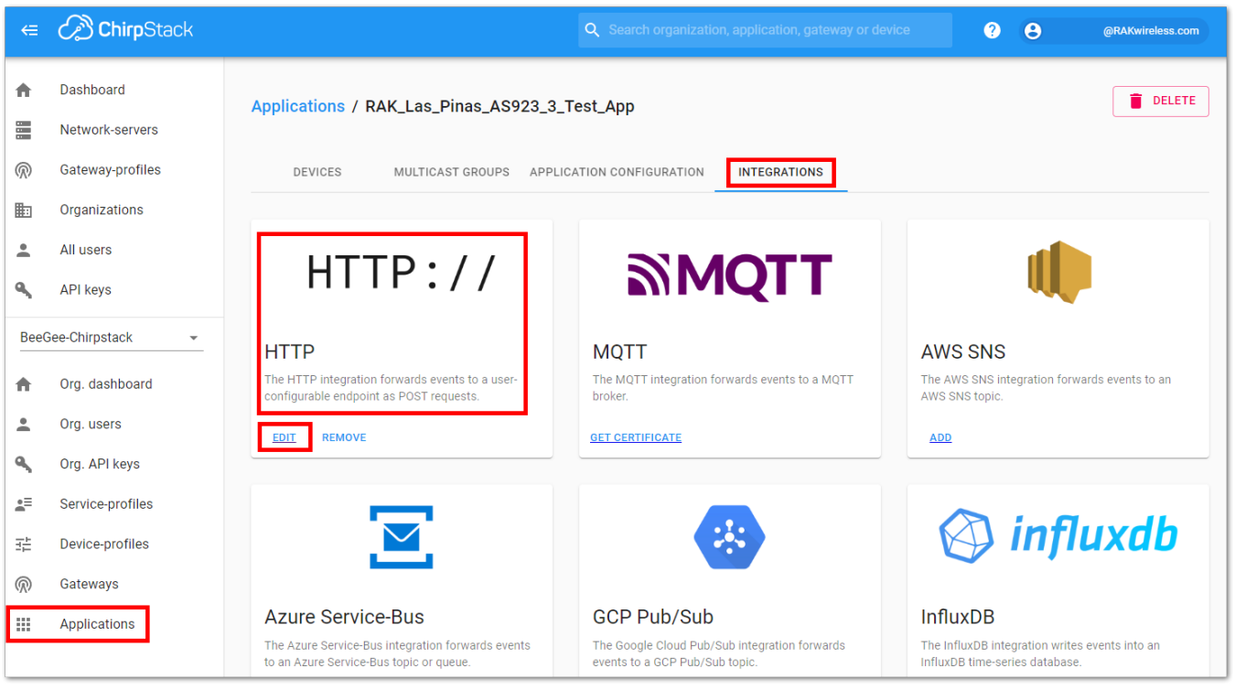

Figure 60: Creating integration

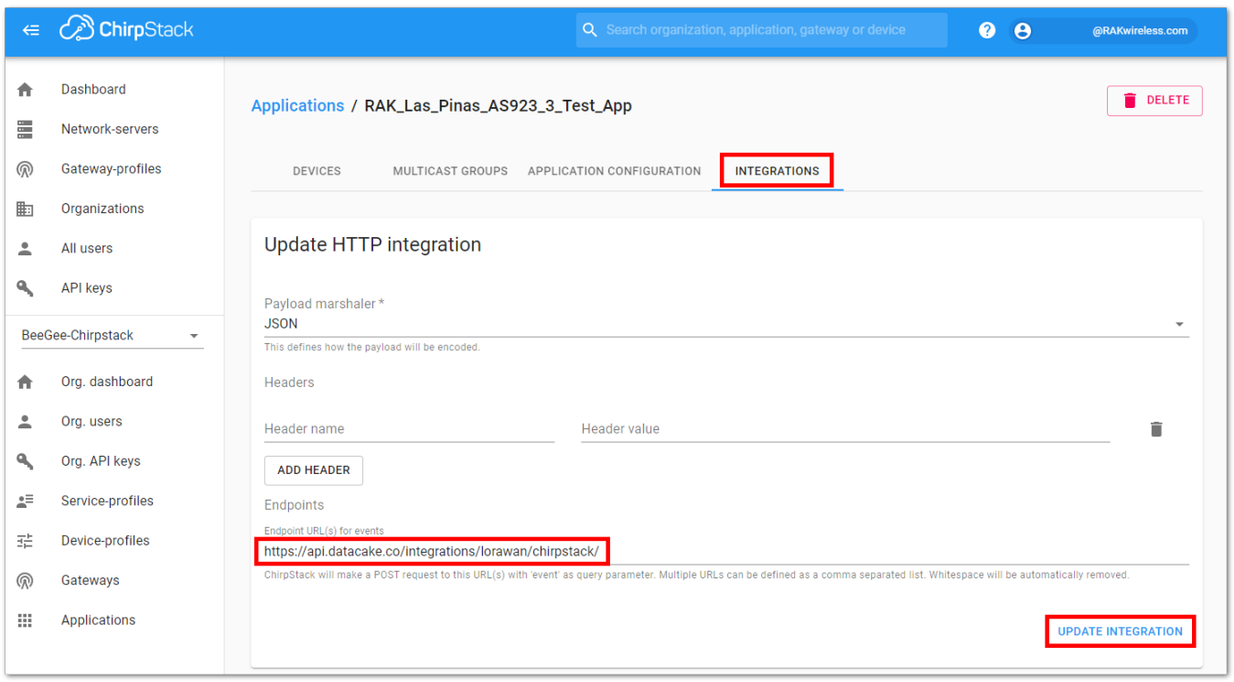

Figure 61: Creating endpoint for Datacake.co

7. The last step on the Chirpstack side, is the creation of the API key. This is needed to allow Datacake in sending downlink packets to the RAK10701 Field Tester. Make sure the key is copied and saved somewhere, it is only retrievable during the key creation. Copy the Token and save it in a text editor.

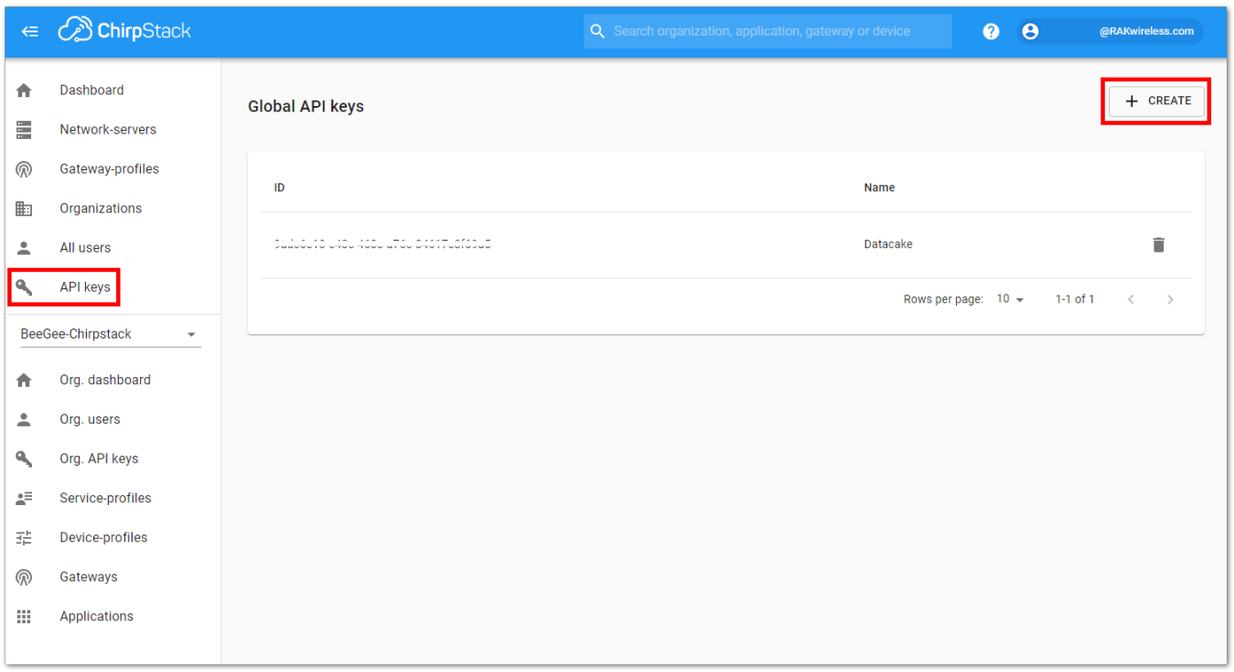

Figure 62: Creation of API Key

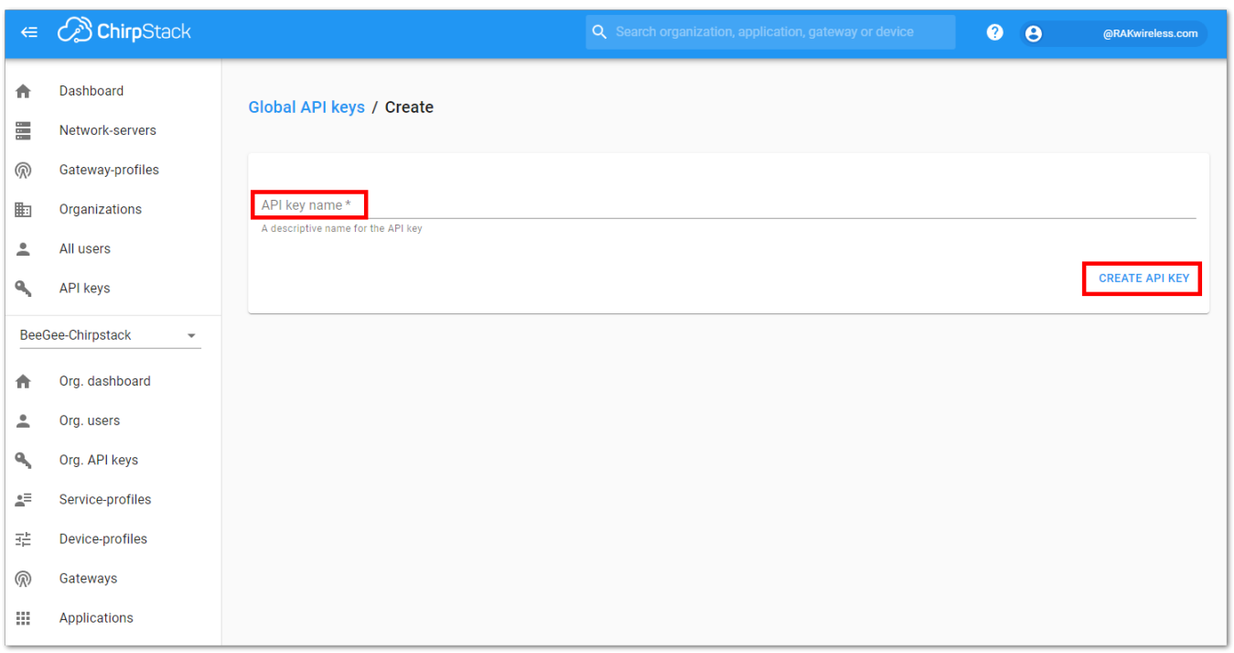

Figure 63: Creation of API Key

8. You can now proceed on Datacake and add a LoRaWAN device that will be linked to your RAK10701 created in ChirpStack. You have to create an account if you do not have one yet.

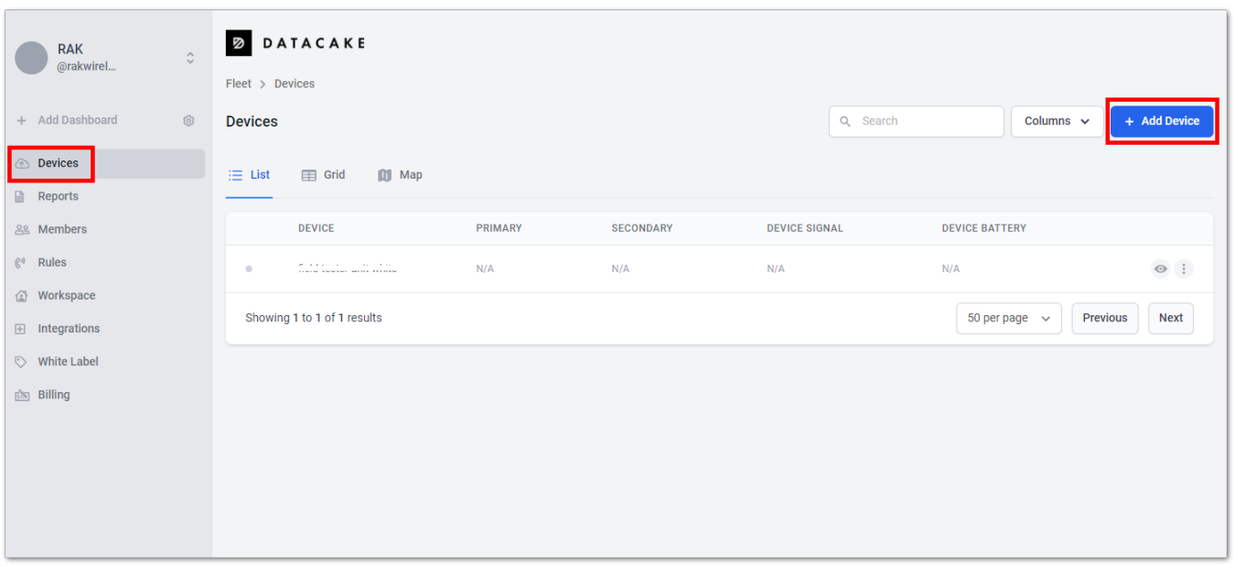

Figure 64: Device list dashboard

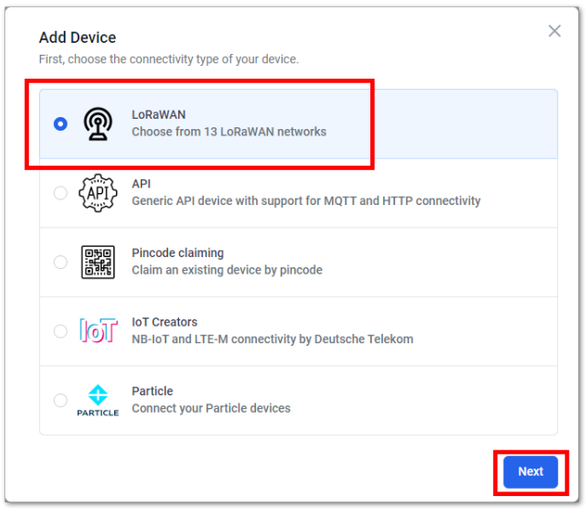

Figure 65: Selecting LoRaWAN

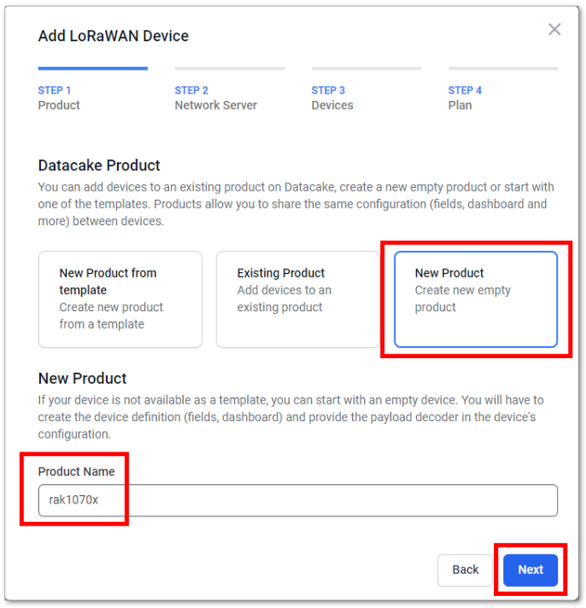

Figure 66: Add new product

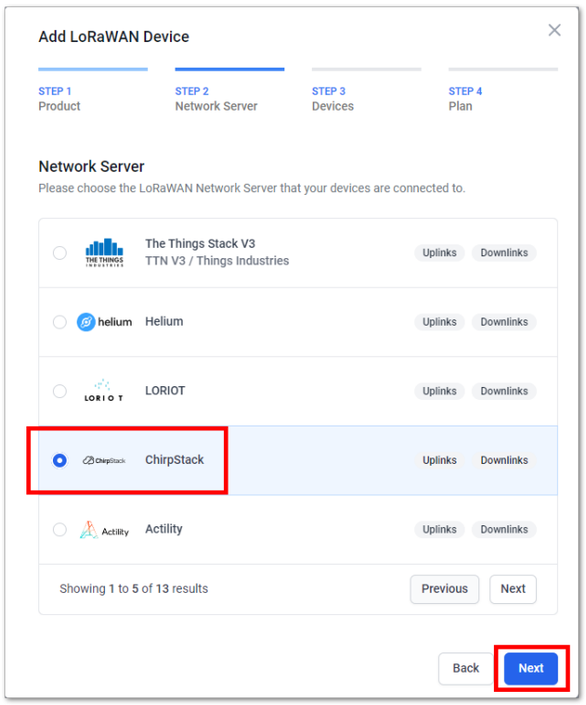

Figure 67: Selecting chirpstack

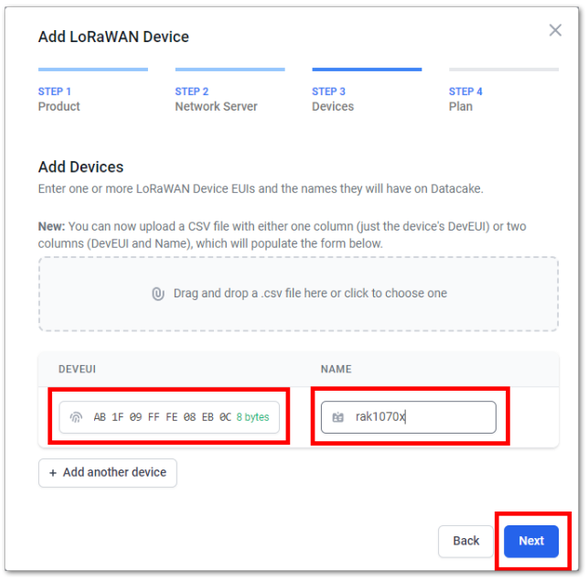

Figure 68: Adding device name

9. Next step is to enable the downlink. This is the step where you'll be needing the previously created API key in step 7 of this guide. Take note that the `ChirpStack URL` should be based on your deployed Chirpstack network server. After doing all configurations, click update and save.

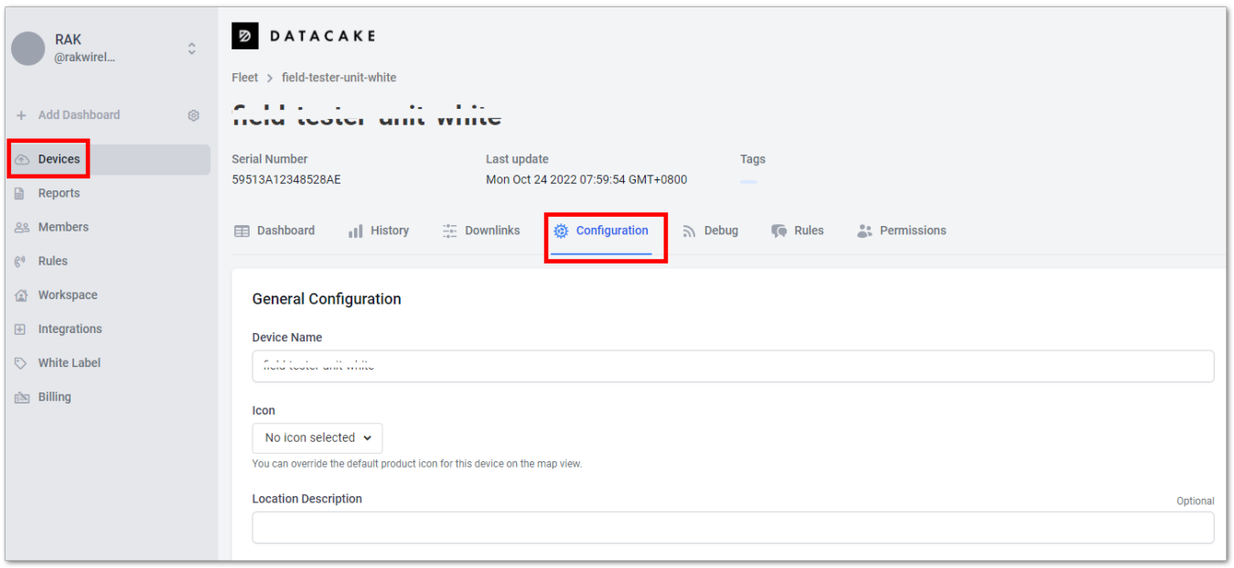

Figure 69: Configuration settings

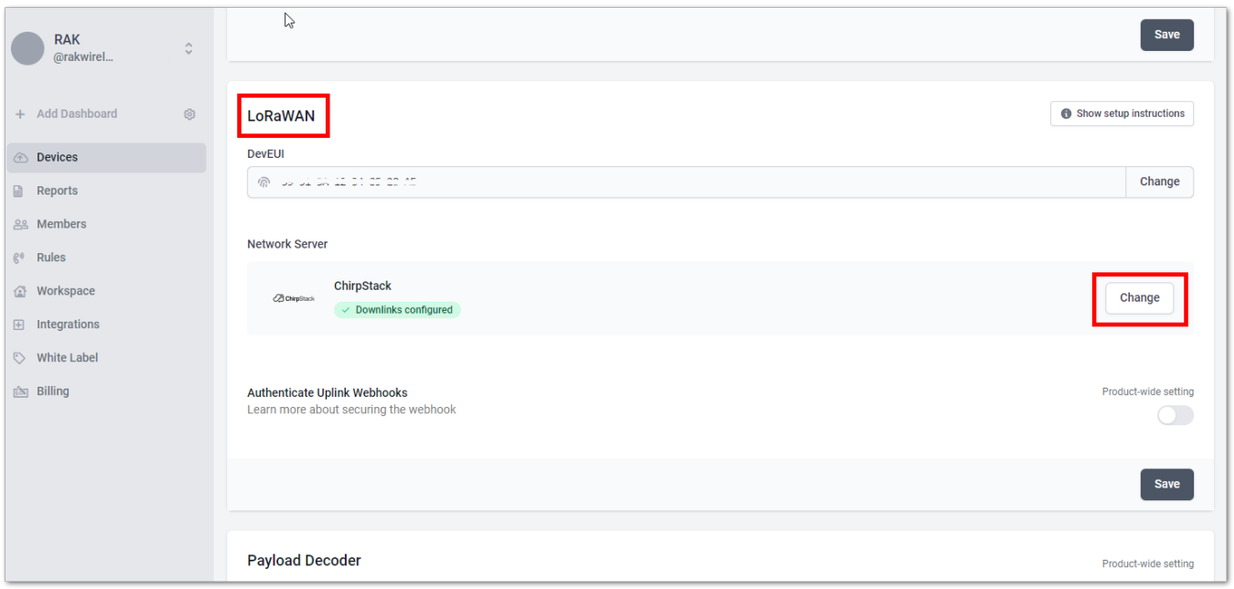

Figure 70: Chirpstack downlink configuration

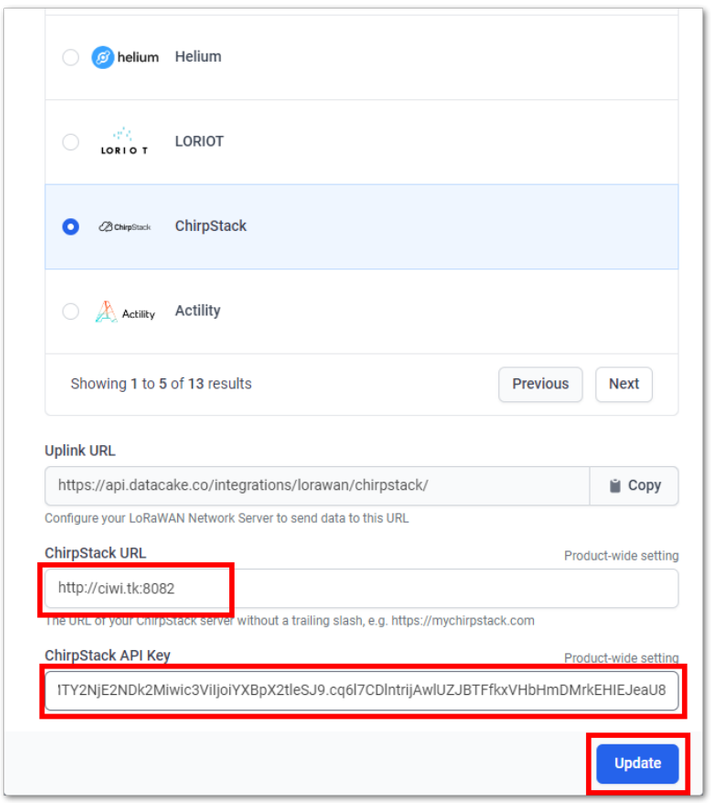

Figure 71: Chirpstack URL and API key

10. Just below the `LoRaWAN` section in datacake.co, you'll see the `Payload Decoder`. This is a very critical step to ensure that all important data will be covered.

```js
function distance(lat1, lon1, lat2, lon2) {
    if ((lat1 == lat2) && (lon1 == lon2)) {
        return 0;
    }
    else {
        var radlat1 = Math.PI * lat1 / 180;
        var radlat2 = Math.PI * lat2 / 180;
        var theta = lon1 - lon2;
        var radtheta = Math.PI * theta / 180;
        var dist = Math.sin(radlat1) * Math.sin(radlat2) + Math.cos(radlat1) * Math.cos(radlat2)
        if (dist > 1) {
            dist = 1;
        }
        dist = Math.acos(dist);
        dist = dist * 180 / Math.PI;
        dist = dist * 60 * 1.1515;
        dist = dist * 1.609344;
        return dist;
    }
}

function Decoder(bytes, fPort) {
    var decoded = {};
    // avoid sending Downlink ACK to integration (Cargo)
    if (fPort === 1) {
        var lonSign = (bytes[0] >> 7) & 0x01 ? -1 : 1;
        var latSign = (bytes[0] >> 6) & 0x01 ? -1 : 1;

        var encLat = ((bytes[0] & 0x3f) << 17) +
            (bytes[1] << 9) +
            (bytes[2] << 1) +
            (bytes[3] >> 7);

        var encLon = ((bytes[3] & 0x7f) << 16) +
            (bytes[4] << 8) +
            bytes[5];

        var hdop = bytes[8] / 10;
        var sats = bytes[9];

        var maxHdop = 2;
        var minSats = 5;

        if ((hdop < maxHdop) && (sats >= minSats)) {
            // Send only acceptable quality of position to mappers
            decoded.latitude = latSign * (encLat * 108 + 53) / 10000000;
            decoded.longitude = lonSign * (encLon * 215 + 107) / 10000000;
            decoded.altitude = ((bytes[6] << 8) + bytes[7]) - 1000;
            decoded.accuracy = (hdop * 5 + 5) / 10
            decoded.hdop = hdop;
            decoded.sats = sats;
            decoded.location = "(" + decoded.latitude + "," + decoded.longitude + ")";
        } else {
            decoded.error = "Need more GPS precision (hdop must be <" + maxHdop +
                " & sats must be >= " + minSats + ") current hdop: " + hdop + " & sats:" + sats;
            decoded.latitude = latSign * (encLat * 108 + 53) / 10000000;
            decoded.longitude = lonSign * (encLon * 215 + 107) / 10000000;
            decoded.altitude = ((bytes[6] << 8) + bytes[7]) - 1000;
            decoded.accuracy = (hdop * 5 + 5) / 10
            decoded.hdop = hdop;
            decoded.sats = sats;
            decoded.location = "(" + decoded.latitude + "," + decoded.longitude + ")";
```

```
}
//      decoded.raw = rawPayload.uplink_message.rx_metadata[0].location;
decoded.num_gw = normalizedPayload.gateways.length;
decoded.minRSSI = 0;
decoded.maxRSSI = 0;
decoded.minSNR = 0;
decoded.maxSNR = 0;
decoded.minDistance = 0;
decoded.maxDistance = 0;

var server_type = 0;
// Check if payload comes from TTN
if (typeof (rawPayload.uplink_message) != "undefined") {
    console.log("Found TTN format");
    server_type = 1;
}
// Check if payload comes from Helium
else if (typeof (rawPayload.hotspots) != "undefined") {
    console.log("Found Helium format");
    server_type = 2;
}
// Check if payload comes from Chirpstack
else if (typeof (rawPayload.rxInfo) != "undefined") {
    console.log("Found Chirpstack format");
    server_type = 3;
    decoded.is_chirpstack = 1;
}
else {
    console.log("Unknown raw format");
}

var gw_lat = {};
var gw_long = {};

decoded.num_gw = 0;
for (idx_tst = 0; idx_tst < 10; idx_tst++)
{
    if (typeof (normalizedPayload.gateways[idx_tst]) != "undefined")
    {
        console.log("Found gateway with IDX " + idx_tst);
        decoded.num_gw += 1;
    }
}

for (idx = 0; idx < decoded.num_gw; idx++) {
    var new_rssi = (!!normalizedPayload.gateways && !!normalizedPayload.gateways[idx] &&
    var new_snr = (!!normalizedPayload.gateways && !!normalizedPayload.gateways[idx] && n
    if ((new_rssi < decoded.minRSSI) || (decoded.minRSSI == 0)) {
        decoded.minRSSI = new_rssi;
    }
    if ((new_rssi > decoded.maxRSSI) || (decoded.maxRSSI == 0)) {
        decoded.maxRSSI = new_rssi;
    }
    if ((new_snr < decoded.minSNR) || (decoded.minSNR == 0)) {
        decoded.minSNR = new_snr;
    }
    if ((new_snr > decoded.maxSNR) || (decoded.maxSNR == 0)) {
        decoded.maxSNR = new_snr;
    }

    // var gw_lat = 0.0;
    // var gw_long = 0.0;
    switch (server_type) {
```

```
            //TTN
            case 1:
                gw_lat[idx] = rawPayload.uplink_message.rx_metadata[idx].location.latitude;
                gw_long[idx] = rawPayload.uplink_message.rx_metadata[idx].location.longitude;
                break;
            // Helium
            case 2:
                gw_lat[idx] = rawPayload.hotspots[idx].lat;
                gw_long[idx] = rawPayload.hotspots[idx].long;
                break;
            // Chirpstack
            case 3:
                gw_lat[idx] = rawPayload.rxInfo[idx].location.latitude;
                gw_long[idx] = rawPayload.rxInfo[idx].location.longitude;
                break;
            default:
                console.log("Unknown LNS");
                break;
        }

        console.log("IDX " + idx + " lat " + gw_lat[idx] + " long " + gw_long[idx]);
        // decoded.gw_lat[idx] = gw_lat;
        // decoded.gw_long[idx] = gw_long;

        // Calculate distance
        var new_distance = distance(gw_lat[idx], gw_long[idx], decoded.latitude, decoded.long
        if ((new_distance < decoded.minDistance) || (decoded.minDistance == 0)) {
            decoded.minDistance = new_distance * 1000;
        }
        if ((new_distance > decoded.maxDistance) || (decoded.maxDistance == 0)) {
            decoded.maxDistance = new_distance * 1000;
        }
    }

    switch (decoded.num_gw) {
        case 20:
            decoded.hotspot_10 = "(" + gw_lat[19] + "," + gw_long[19] + ")";
        case 19:
            decoded.hotspot_09 = "(" + gw_lat[18] + "," + gw_long[18] + ")";
        case 18:
            decoded.hotspot_08 = "(" + gw_lat[17] + "," + gw_long[17] + ")";
        case 17:
            decoded.hotspot_07 = "(" + gw_lat[16] + "," + gw_long[16] + ")";
        case 16:
            decoded.hotspot_06 = "(" + gw_lat[15] + "," + gw_long[15] + ")";
        case 15:
            decoded.hotspot_05 = "(" + gw_lat[14] + "," + gw_long[14] + ")";
        case 14:
            decoded.hotspot_04 = "(" + gw_lat[13] + "," + gw_long[13] + ")";
        case 13:
            decoded.hotspot_03 = "(" + gw_lat[12] + "," + gw_long[12] + ")";
        case 12:
            decoded.hotspot_02 = "(" + gw_lat[11] + "," + gw_long[11] + ")";
        case 11:
            decoded.hotspot_01 = "(" + gw_lat[10] + "," + gw_long[10] + ")";
        case 10:
            decoded.hotspot_10 = "(" + gw_lat[9] + "," + gw_long[9] + ")";
        case 9:
            decoded.hotspot_09 = "(" + gw_lat[8] + "," + gw_long[8] + ")";
        case 8:
            decoded.hotspot_08 = "(" + gw_lat[7] + "," + gw_long[7] + ")";
        case 7:
            decoded.hotspot_07 = "(" + gw_lat[6] + "," + gw_long[6] + ")";
```

```
            case 6:
                decoded.hotspot_06 = "(" + gw_lat[5] + "," + gw_long[5] + ")";
            case 5:
                decoded.hotspot_05 = "(" + gw_lat[4] + "," + gw_long[4] + ")";
            case 4:
                decoded.hotspot_04 = "(" + gw_lat[3] + "," + gw_long[3] + ")";
            case 3:
                decoded.hotspot_03 = "(" + gw_lat[2] + "," + gw_long[2] + ")";
            case 2:
                decoded.hotspot_02 = "(" + gw_lat[1] + "," + gw_long[1] + ")";
            case 1:
                decoded.hotspot_01 = "(" + gw_lat[0] + "," + gw_long[0] + ")";
            default:
                break;
        }

        decoded.maxMod = parseInt((decoded.maxDistance / 250), 10);
        decoded.minMod = parseInt((decoded.minDistance / 250), 10);
        decoded.maxDistance = parseInt((decoded.maxMod * 250), 10);
        decoded.minDistance = parseInt((decoded.minMod * 250), 10);
        if (decoded.maxDistance <= 1) {
            decoded.maxDistance = parseInt(250, 10);
        }
        if (decoded.minDistance <= 1) {
            decoded.minDistance = parseInt(250, 10);
        }
        return decoded;
    }
    return null;

}
```

This decoder is not only decoding data from the LoRaWAN packet but is as well reading gateway information from the additional data that the LoRaWAN server added to the data it forwarded to Datacake.

Each LoRaWAN server uses a different format for this additional information, so there is a code section that tries to detect whether the data came from a Chirpstack LSN, from TTN, or from a Helium Console:

```js
var server_type = 0;
        // Check if payload comes from TTN
        if (typeof (rawPayload.uplink_message) != "undefined") {
            console.log("Found TTN format");
            server_type = 1;
        }
        // Check if payload comes from Helium
        else if (typeof (rawPayload.hotspots) != "undefined") {
            console.log("Found Helium format");
            server_type = 2;
        }
        // Check if payload comes from Chirpstack
        else if (typeof (rawPayload.rxInfo) != "undefined") {
            console.log("Found Chirpstack format");
            server_type = 3;
            decoded.is_chirpstack = 1;
        }
        else {
            console.log("Unknown raw format");
        }
```

Once the data is extracted, it calculates the distance between the RAK10701 Field Tester location and the different gateways that received the LoRaWAN packet. This version of the decoder can handle up to 10 gateways, but it can be extended.

```js
function distance(lat1, lon1, lat2, lon2) {
    if ((lat1 == lat2) && (lon1 == lon2)) {
        return 0;
    }
    else {
        var radlat1 = Math.PI * lat1 / 180;
        var radlat2 = Math.PI * lat2 / 180;
        var theta = lon1 - lon2;
        var radtheta = Math.PI * theta / 180;
        var dist = Math.sin(radlat1) * Math.sin(radlat2) + Math.cos(radlat1) * Math.cos(radlat2)
        if (dist > 1) {
            dist = 1;
        }
        dist = Math.acos(dist);
        dist = dist * 180 / Math.PI;
        dist = dist * 60 * 1.1515;
        dist = dist * 1.609344;
        return dist;
    }
}
```

In the next step, it analyzes the different distances and RSSI levels to find the closest and farthest gateway and the lowest and highest RSSI and SNR levels.

The result of the decoding is then put into different data fields that are used by Chirpstack for the visualization and by the rule, we will define to create the downlink to the RAK10701 Field Tester.

11. The next step is to create the different data fields that are filled by the data decoder. This is done in the Fields section of the device configuration, just below the data encoder section.

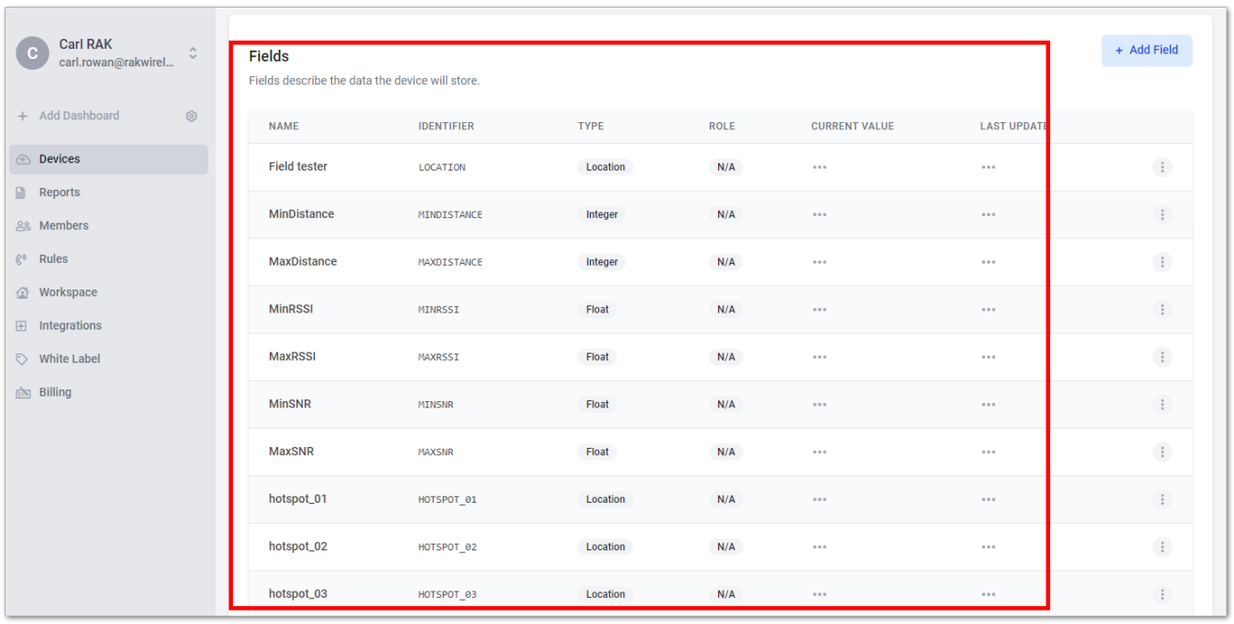

Figure 72: Creation of data field

The following fields are required:

Field Name	Identifier (returned value from the decoder)	Field Type
Field Tester	LOCATION	Location
MinDistance	MINDISTANCE	Integer
MaxDistance	MAXDISTANCE	Integer
MinRSSI	MINRSSI	Float
MaxRSSI	MAXRSSI	Float
MinSNR	MINSNR	Float
MaxSNR	MAXSNR	Float
minMod	MINMOD	Integer
maxMod	MAXMAD	Integer
hotspot_01	HOTSPOT_01	Location
hotspot_02	HOTSPOT_01	Location
hotspot_0…	HOTSPOT_...	Location
hotspot_09	HOTSPOT_09	Location
hotspot_10	HOTSPOT_10	Location
is_chirpstack	IS_CHIRPSTACK	Boolean

There are more variables created by the decoder, but this is the minimum set required for the dashboard and to configure the download.

12. This is the most critical step so that the RAK10701 will be able to display the necessary details helpful in Field Testing the LoRaWAN network. This section is responsible for the "backend-server functions". In this step, we create the automatic downlink to the device that is executed every time a data packet from the RAK10701 Field Tester arrives. Take note that port number 2 is used by RAK10701 for downlinks and `Trigger on measurements` should be checked.

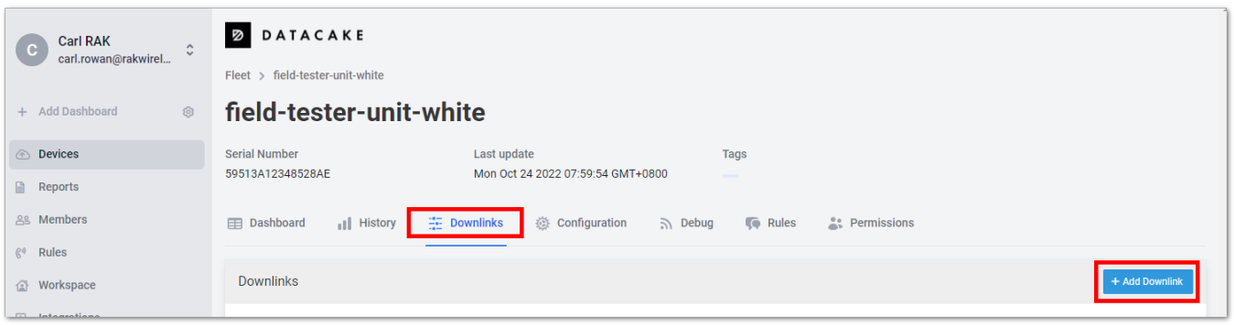

Figure 73: Chirpstack downlink configuration

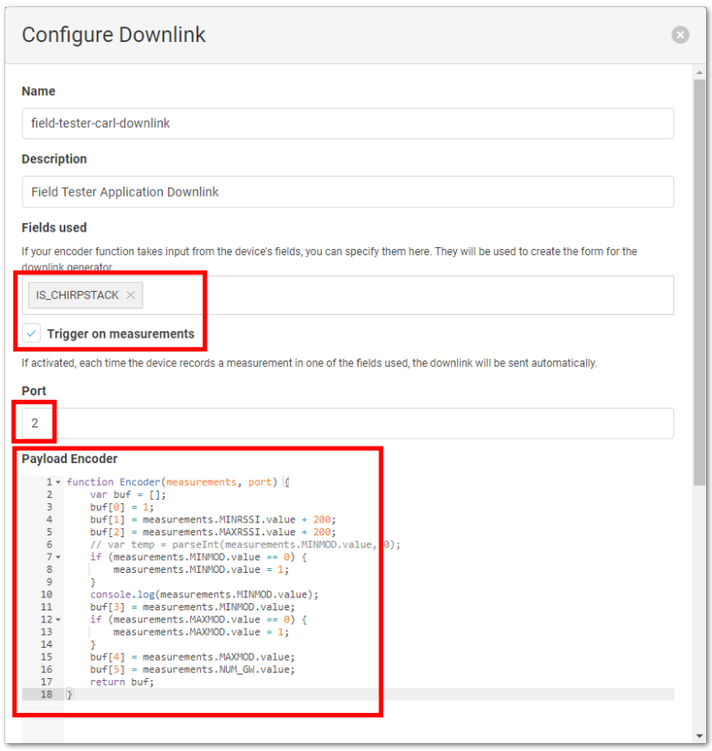

Figure 74: Chirpstack downlink configuration

In this guide, the downlink is only created when the Field Tester is connected through a Chirpstack server. When TTN or Helium is used, the downlink would be created by the original backend server.

This is controlled by the “Fields used” where you can see the IS_CHIRPSTACK.

The Payload Encoder is preparing a downlink packet that will be sent back to the RAK10701 Field Tester. The downlink packet is only 6 bytes large to avoid problems in LoRaWAN regions with limited downlink packet sizes.

This is the complete decoder script. The first byte is usually a counter, but it works well if the counter is ignored and always set to 1.

```js
function Encoder(measurements, port) {
    var buf = [];
    buf[0] = 1;
    buf[1] = measurements.MINRSSI.value + 200;
    buf[2] = measurements.MAXRSSI.value + 200;
    // var temp = parseInt(measurements.MINMOD.value,10);
    if (measurements.MINMOD.value == 0) {
        measurements.MINMOD.value = 1;
    }
    console.log(measurements.MINMOD.value);
    buf[3] = measurements.MINMOD.value;
    if (measurements.MAXMOD.value == 0) {
        measurements.MAXMOD.value = 1;
    }
    buf[4] = measurements.MAXMOD.value;
    buf[5] = measurements.NUM_GW.value;
    return buf;
}
```

13. You can now proceed on device configuration so that the proper EUIs and KEY will match the one in the network server.

RAK10701-P Field Tester Pro Guide for LORIOT and Datacake

In this document, you will find a step-by-step guide for performing a field mapping test using LORIOT network management system and Datacake's platform to visualize your results. This solution will help you in your network planning ventures and ensure your decisions are data-driven and adequate to your surroundings.

Prerequisites

- RAK10701 WisNode Field Tester for LoRaWAN
- LORIOT account
- Datacake account
- Gateway

Setting LORIOT as the LNS

1. Forward a gateway to LORIOT, which will be the LNS (LoRa Network Server) for this use case. For registration of the gateway to LORIOT, you will need the gateway's MAC and EUI, which can be found on the Overview page of WisGateOS 2.

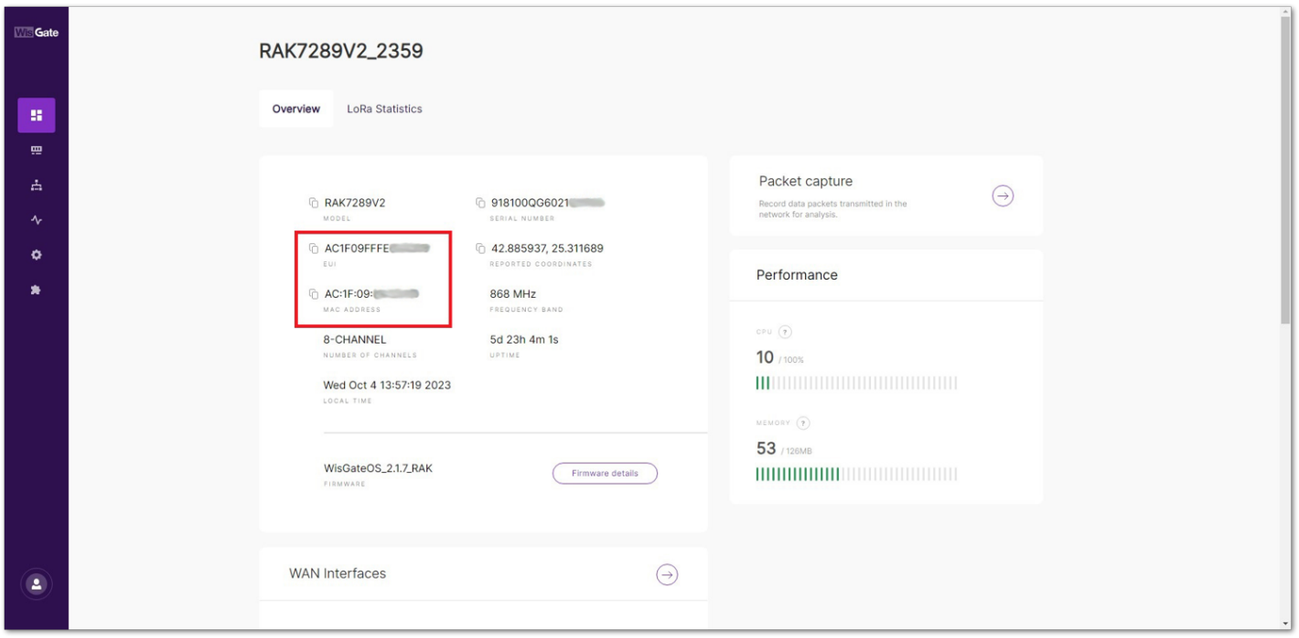

Figure 75: WisGate Edges web UI

2. Go to your LORIOT profile. From the menu on the left, navigate through **Networks>{your_network}>+Add Gateway**.

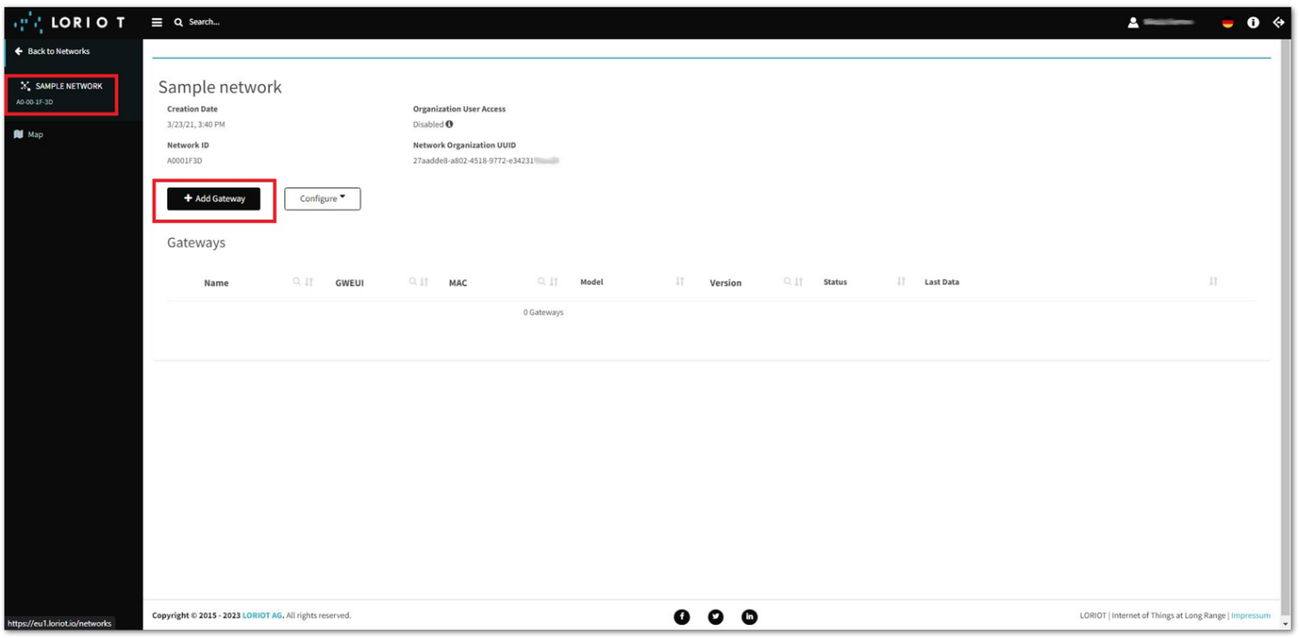

Figure 76: LORIOT console

> 📝 **NOTE**
>
> The LORIOT platform provides you with a **Sample Network** at the point of your profile creation. You can use it for free. If you wish to create a new one, or delete the provided one, you will need a paid plan to continue.

3. For the base platform select **Basics Station Semtech**. You will be asked to provide eth0 MAC address and EUI, which you obtained in step 1. After filling in these values, press the **Register Basics Station Semtech gateway** at the bottom of the page.

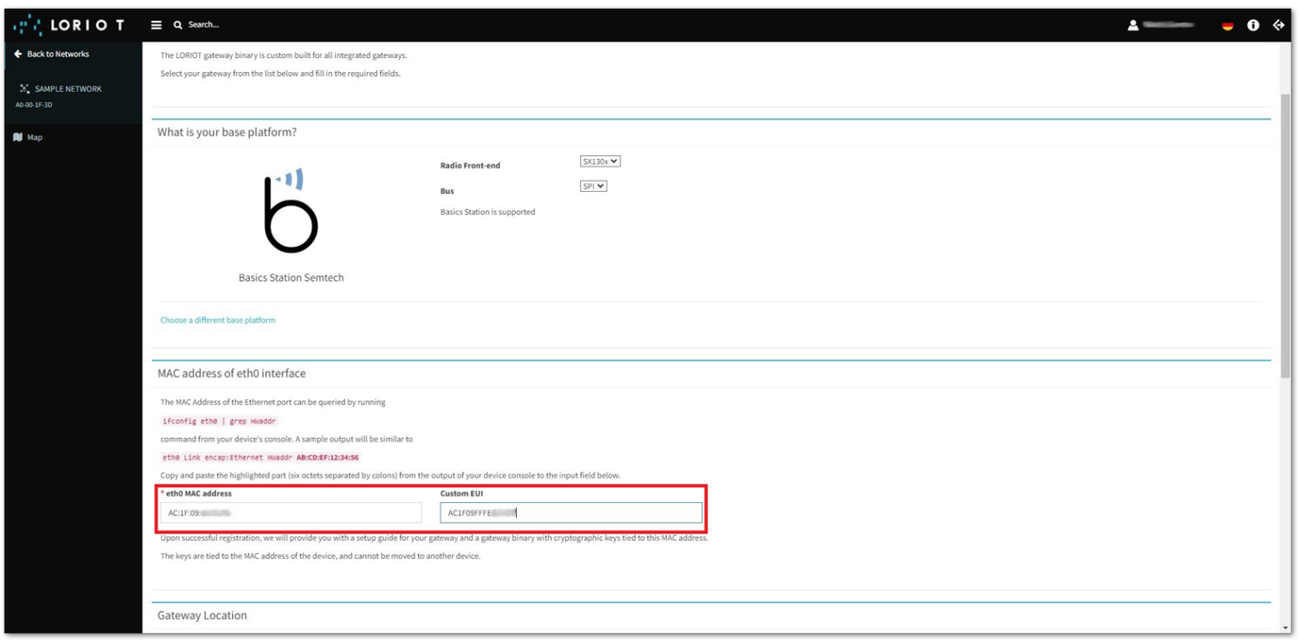

Figure 77: Registering The Gateway To LORIOT

4. The last thing you need to do to connect your gateway with LORIOT LNS is to provide the Basics Station configuration to the gateway. This can be done by going to the gateway's **web UI**>**LoRa**>**Configuration** and doing a Basics station server setup.

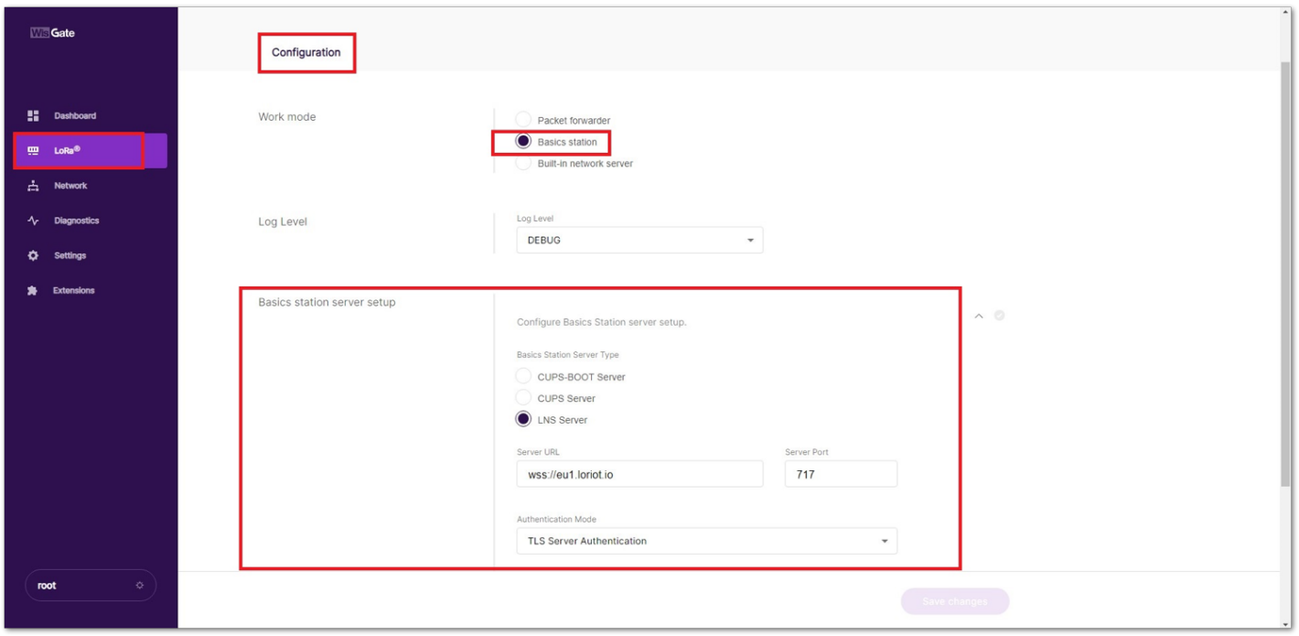

Figure 78: Gateway Configuration Page

You can find the Trust (CA Certificate), the Server URL, and the Server port in LORIOT by navigating to the newly registered **Gateway**>**Certificate**. Use the configuration provided by LORIOT as it may differ from the guide depending on your region.

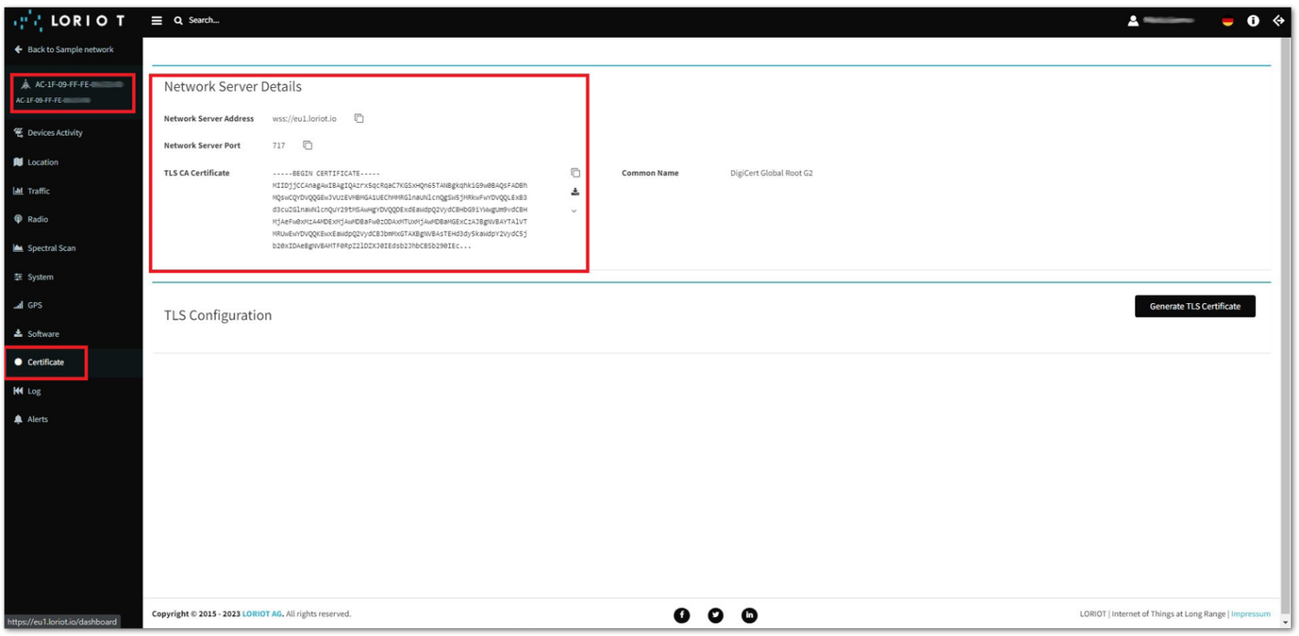

Figure 79: Certificate

5. If the steps are followed correctly, the gateway should show a **Connected** status.

Adding the Device and LORIOT to Datacake Integration

1. Add the device to LORIOT. In the LORIOT platform, navigate to **Applications**>**{your_appliaction}** and use the **Enroll Device** utility from the menu on the left. Fill out your Device EUI, Join (APP) EUI, and Application Key.

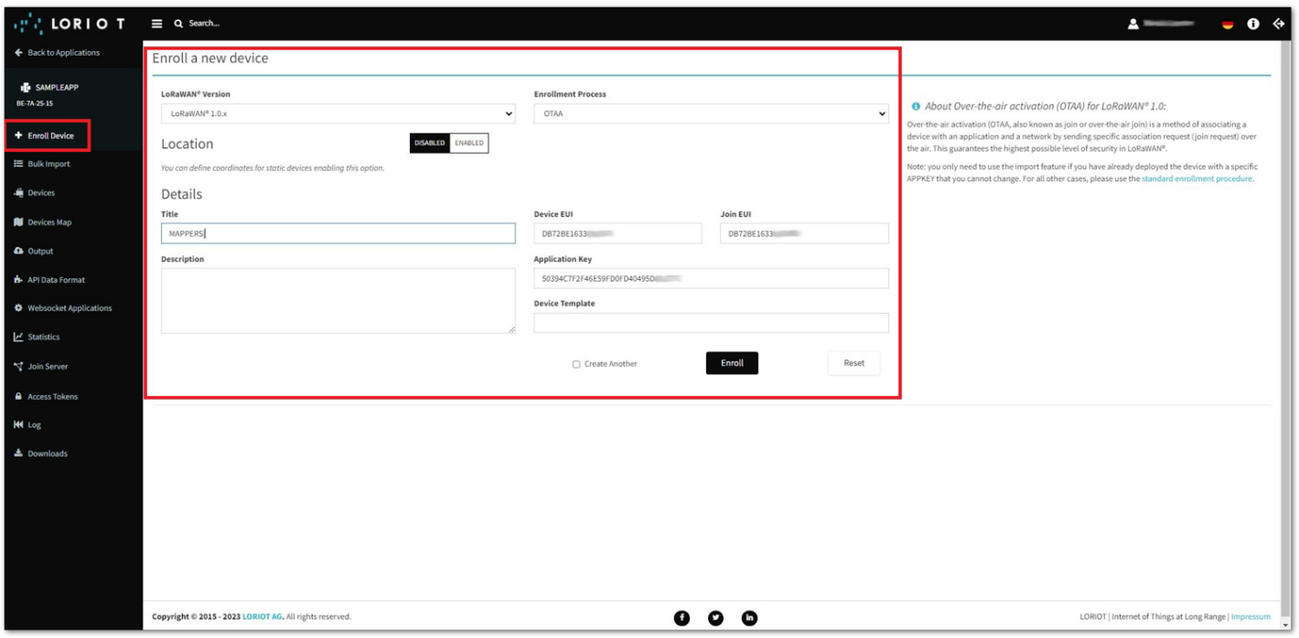

Figure 80: Adding The Device To The LORIOT Platform

> 📝 **NOTE**
>
> The LORIOT platform provides you with a **Sample Application** at the point of your profile creation. You can use it for free. If you wish to create a new one or delete the provided one, you will need a paid plan to continue.

2. Use the Output utility to set up the Datacake integration. For now, just give it a name.The Authorization requires additional settings that will not be covered by this guide. For more information regarding this process, refer to Datacake's guide .

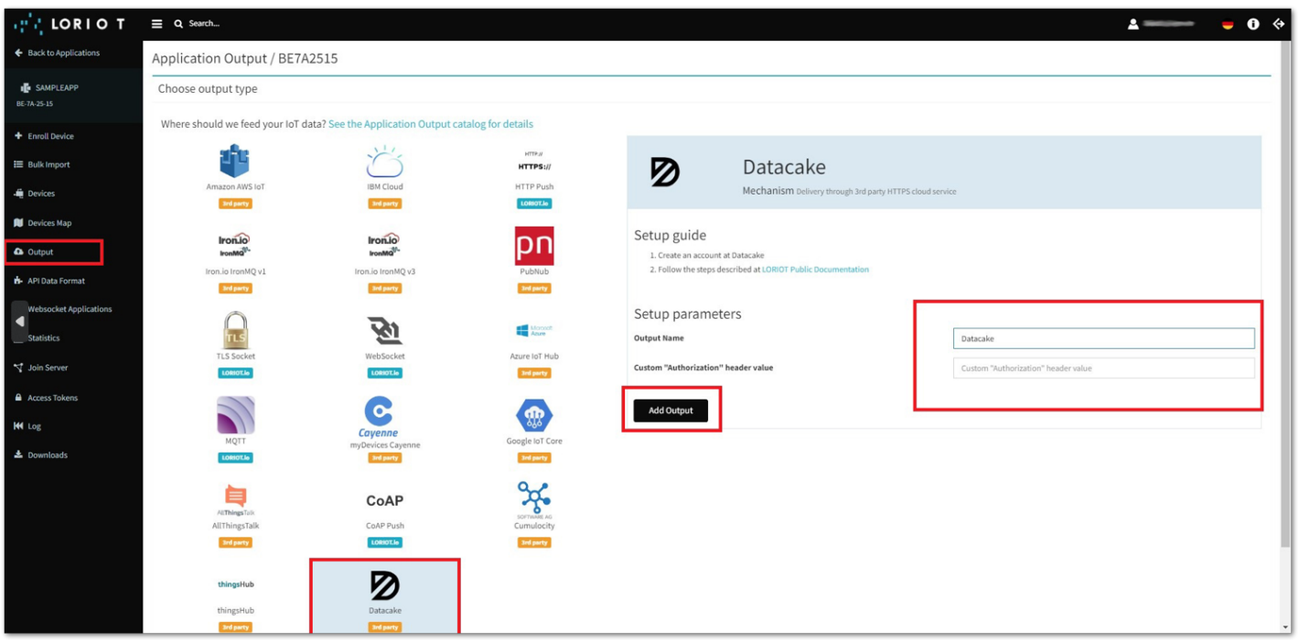

Figure 81: Add Output

3. Now, you need to add the device in Datacake. To register a new device, navigate to the **Devices** tab in your Datacake account. Click the **+Add Device** button.

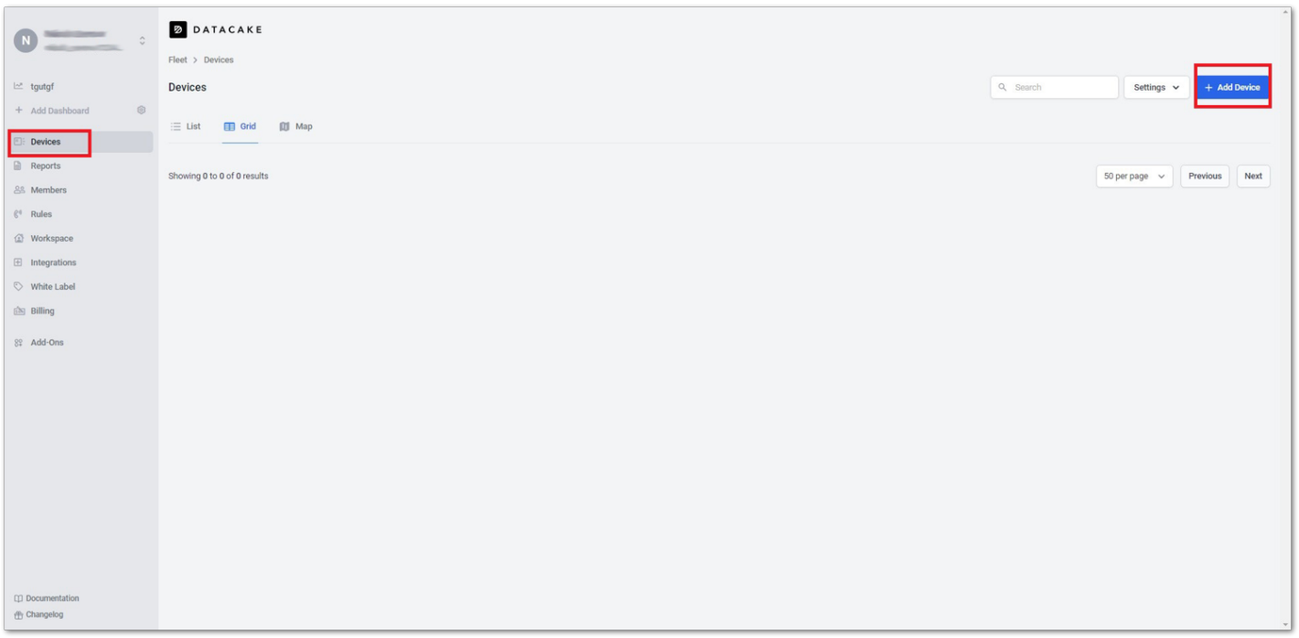

Figure 82: Datacake Platform

4. Choose **New Product** under **Datacake Product**. Enter the device name in the **Product name** input box, and proceed by clicking **Next**.

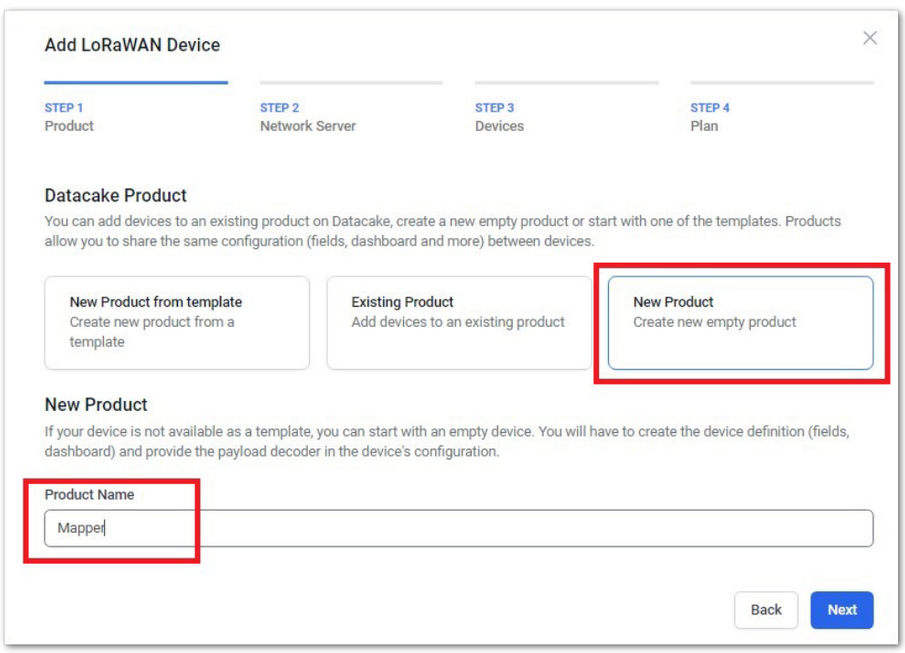

Figure 83: Add Lorawan Device 1

5. Select **LORIOT** as the Network Server and click **Next**.

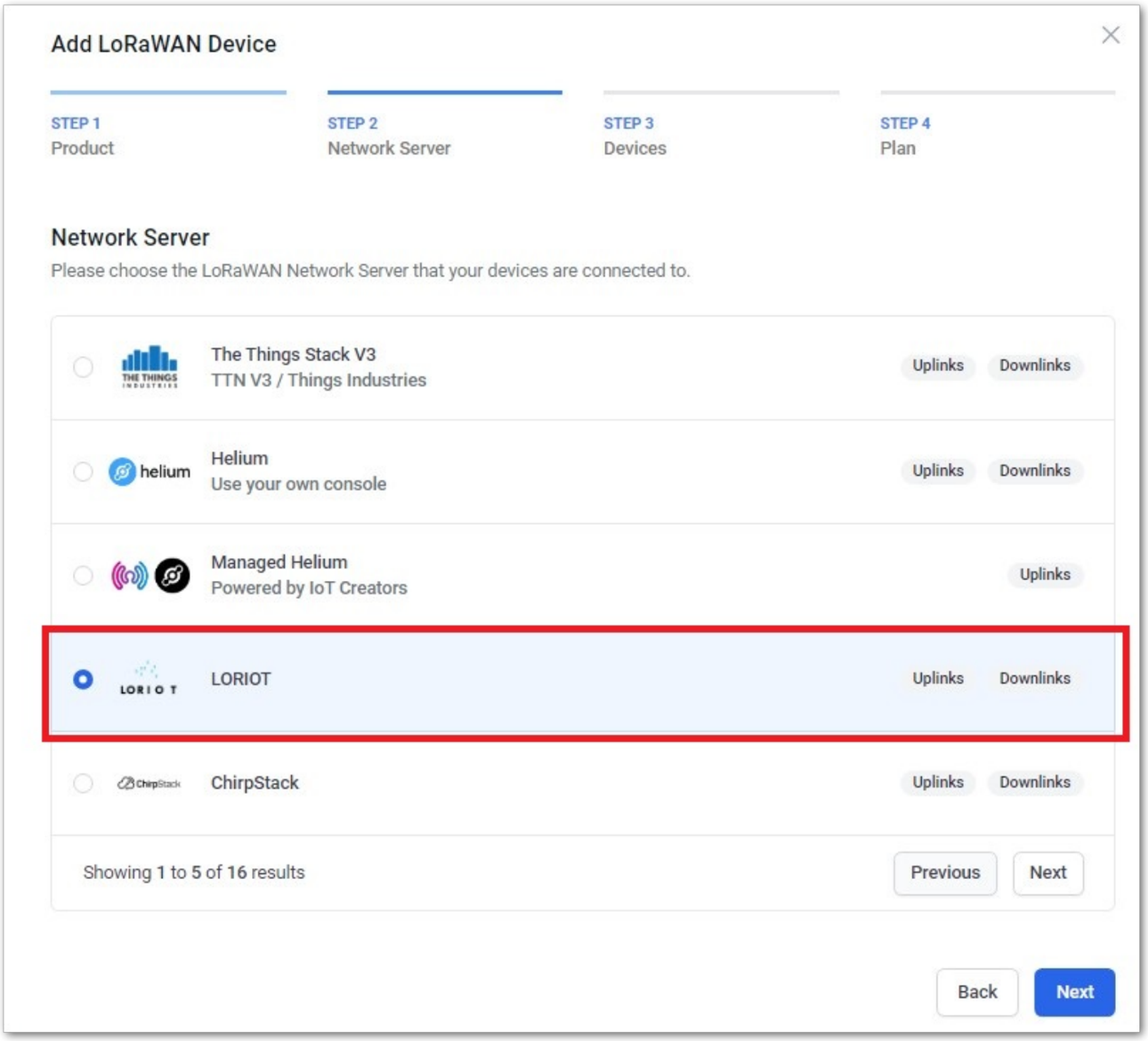

Figure 84: Add Lorawan Device 2

6. On the next page, you will have to enter the name and DEVEUI of the device. Select the plan for Datacake according to your needs and finish the device-adding procedure.

7. Now, navigate to **Configuration** in the newly created device at Datacake and scroll down to the **Network Server** configuration. Click **Change**.

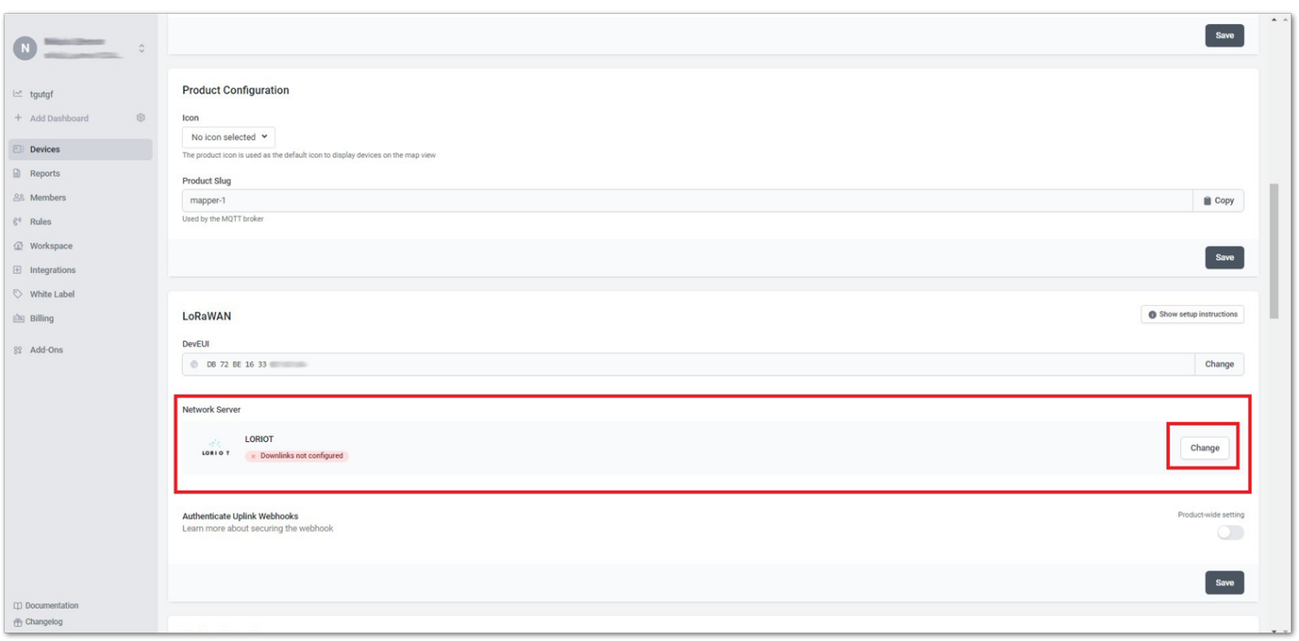

Figure 85: Device Network Server Configuration

8. You will need the LORIOT Access Token, which is generated from the LORIOT console. Navigate to **LORIOT**>**Access Tokens** and copy the token to put it in Datacake.

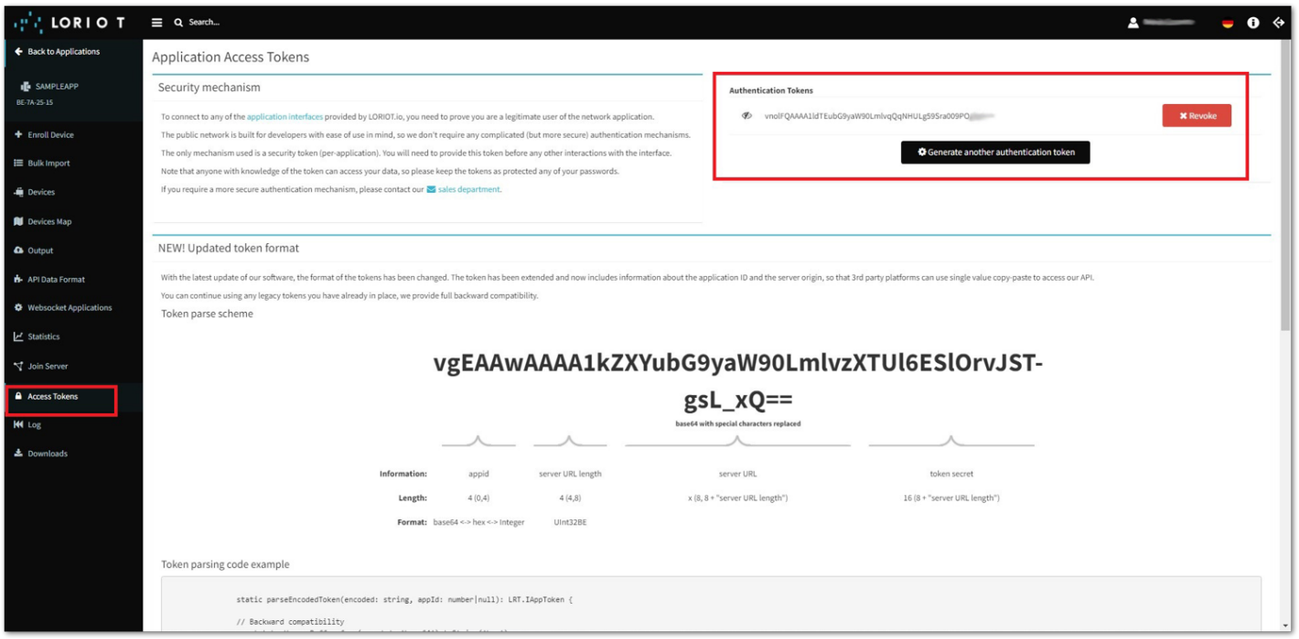

Figure 86: LORIOT Access Token

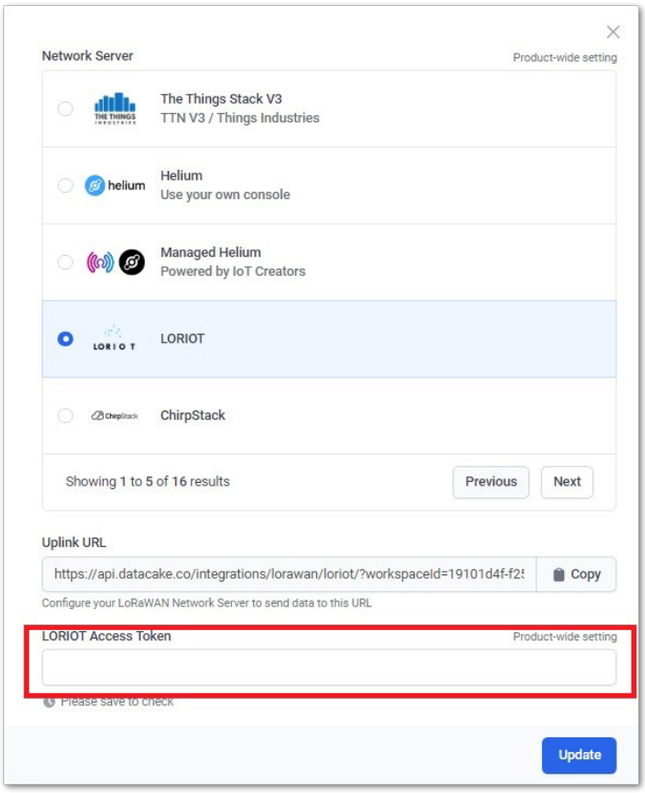

Figure 87: Access Token Field In Datacake

You can generate a new access token or use the existing one.

9. If everything is done correctly, you should see a trickle of RAW data in the Debug window of Datacake.

Setting up the Uplink Payload Decoder and the Downlink Encoder

1. In the Datacake console, navigate to the **Configuration** tab of your RAK10701 device. At the bottom of the page, you will find the **Payload Decoder** field. Copy the decoder provided below and paste it in that field. This decoder will calculate the values displayed on the screen of the Field Tester.

```js
function distance(lat1, lon1, lat2, lon2) {
    if ((lat1 == lat2) && (lon1 == lon2)) {
        return 0;
    }
    else {
        var radlat1 = Math.PI * lat1 / 180;
        var radlat2 = Math.PI * lat2 / 180;
        var theta = lon1 - lon2;
        var radtheta = Math.PI * theta / 180;
        var dist = Math.sin(radlat1) * Math.sin(radlat2) + Math.cos(radlat1) * Math.cos(radlat2)
        if (dist > 1) {
            dist = 1;
        }
        dist = Math.acos(dist);
        dist = dist * 180 / Math.PI;
        dist = dist * 60 * 1.1515;
        dist = dist * 1.609344;
        return dist;
    }
}

function Decoder(bytes, fPort) {
    var decoded = {};
    // avoid sending Downlink ACK to integration (Cargo)
    if (fPort === 1) {
        var lonSign = (bytes[0] >> 7) & 0x01 ? -1 : 1;
        var latSign = (bytes[0] >> 6) & 0x01 ? -1 : 1;

        var encLat = ((bytes[0] & 0x3f) << 17) +
            (bytes[1] << 9) +
            (bytes[2] << 1) +
            (bytes[3] >> 7);

        var encLon = ((bytes[3] & 0x7f) << 16) +
            (bytes[4] << 8) +
            bytes[5];

        var hdop = bytes[8] / 10;
        var sats = bytes[9];

        var maxHdop = 2;
        var minSats = 5;

        if ((hdop < maxHdop) && (sats >= minSats)) {
            // Send only acceptable quality of position to mappers
            decoded.latitude = latSign * (encLat * 108 + 53) / 10000000;
            decoded.longitude = lonSign * (encLon * 215 + 107) / 10000000;
            decoded.altitude = ((bytes[6] << 8) + bytes[7]) - 1000;
            decoded.accuracy = (hdop * 5 + 5) / 10
            decoded.hdop = hdop;
            decoded.sats = sats;
            decoded.location = "(" + decoded.latitude + "," + decoded.longitude + ")";
        } else {
            decoded.error = "Need more GPS precision (hdop must be <" + maxHdop +
                " & sats must be >= " + minSats + ") current hdop: " + hdop + " & sats:" + sats;
            decoded.latitude = latSign * (encLat * 108 + 53) / 10000000;
            decoded.longitude = lonSign * (encLon * 215 + 107) / 10000000;
            decoded.altitude = ((bytes[6] << 8) + bytes[7]) - 1000;
            decoded.accuracy = (hdop * 5 + 5) / 10
            decoded.hdop = hdop;
            decoded.sats = sats;
            decoded.location = "(" + decoded.latitude + "," + decoded.longitude + ")";
```

```
}
//      decoded.raw = rawPayload.uplink_message.rx_metadata[0].location;
decoded.num_gw = normalizedPayload.gateways.length;
decoded.minRSSI = 0;
decoded.maxRSSI = 0;
decoded.minSNR = 0;
decoded.maxSNR = 0;
decoded.minDistance = 0;
decoded.maxDistance = 0;
var server_type = 0;
// Check if payload comes from TTN
if (typeof (rawPayload.uplink_message) != "undefined") {
    console.log("Found TTN format");
    server_type = 1;
    decoded.is_chirpstack = 1;
}
// Check if payload comes from Helium
else if (typeof (rawPayload.hotspots) != "undefined") {
    console.log("Found Helium format");
    server_type = 2;
}
// Check if payload comes from Chirpstack
else if (typeof (rawPayload.rxInfo) != "undefined") {
    console.log("Found Chirpstack format");
    server_type = 3;
    decoded.is_chirpstack = 1;
}
// Check if payload comes from LORIOT
else if (typeof (rawPayload.cmd) != "undefined") {
    console.log("Found LORIOT format");
    server_type = 4;
    decoded.is_chirpstack = 1;
}
else {
    console.log("Unknown raw format");
}

var gw_lat = {};
var gw_long = {};

decoded.num_gw = 0;
for (idx_tst = 0; idx_tst < 10; idx_tst++)
{
    if (typeof (normalizedPayload.gateways[idx_tst]) != "undefined")
    {
        console.log("Found gateway with IDX " + idx_tst);
        decoded.num_gw += 1;
    }
}

for (idx = 0; idx < decoded.num_gw; idx++) {
    var new_rssi = (!!normalizedPayload.gateways && !!normalizedPayload.gateways[idx] &&
    var new_snr = (!!normalizedPayload.gateways && !!normalizedPayload.gateways[idx] && n
    if ((new_rssi < decoded.minRSSI) || (decoded.minRSSI == 0)) {
        decoded.minRSSI = new_rssi;
    }
    if ((new_rssi > decoded.maxRSSI) || (decoded.maxRSSI == 0)) {
        decoded.maxRSSI = new_rssi;
    }
    if ((new_snr < decoded.minSNR) || (decoded.minSNR == 0)) {
        decoded.minSNR = new_snr;
    }
    if ((new_snr > decoded.maxSNR) || (decoded.maxSNR == 0)) {
```

```
                decoded.maxSNR = new_snr;
            }
            switch (server_type) {
                //TTN
                case 1:
                    gw_lat[idx] = rawPayload.uplink_message.rx_metadata[idx].location.latitude;
                    gw_long[idx] = rawPayload.uplink_message.rx_metadata[idx].location.longitude;
                    break;
                // Helium
                case 2:
                    gw_lat[idx] = rawPayload.hotspots[idx].lat;
                    gw_long[idx] = rawPayload.hotspots[idx].long;
                    break;
                // Chirpstack
                case 3:
                    gw_lat[idx] = rawPayload.rxInfo[idx].location.latitude;
                    gw_long[idx] = rawPayload.rxInfo[idx].location.longitude;
                    break;

                //LORIOT
                case 4:
                    gw_lat[idx] = rawPayload.gws[0].lat;
                    gw_long[idx] = rawPayload.gws[0].lon;
                    break;
                default:
                    console.log("Unknown LNS");
                    break;
            }

            console.log("IDX " + idx + " lat " + gw_lat[idx] + " long " + gw_long[idx]);

            // Calculate distance
            var new_distance = distance(gw_lat[idx], gw_long[idx], decoded.latitude, decoded.long
            if ((new_distance * 1000 < decoded.minDistance) || (decoded.minDistance == 0)) {
                decoded.minDistance = new_distance * 1000;
            }
            if ((new_distance * 1000 > decoded.maxDistance) || (decoded.maxDistance == 0)) {
                decoded.maxDistance = new_distance * 1000;
            }
        }

        var hotspot_name = ""
        for (idx = 0; idx < decoded.num_gw; idx++) {
            var index = idx + 1;
            if (index < 9) {
                hotspot_name = "hotspot_0" + index.toString();
            } else {
                hotspot_name = "hotspot_" + index.toString();
            }
            console.log(hotspot_name);
            decoded[hotspot_name] = "(" + gw_lat[idx] + "," + gw_long[idx] + ")";
        }

        decoded.maxMod = 1 + parseInt((Math.round(decoded.maxDistance / 250.0)), 10);
        decoded.minMod = 1 + parseInt((Math.round(decoded.minDistance / 250.0)), 10);

        return decoded;
    }
    return null;

}
```

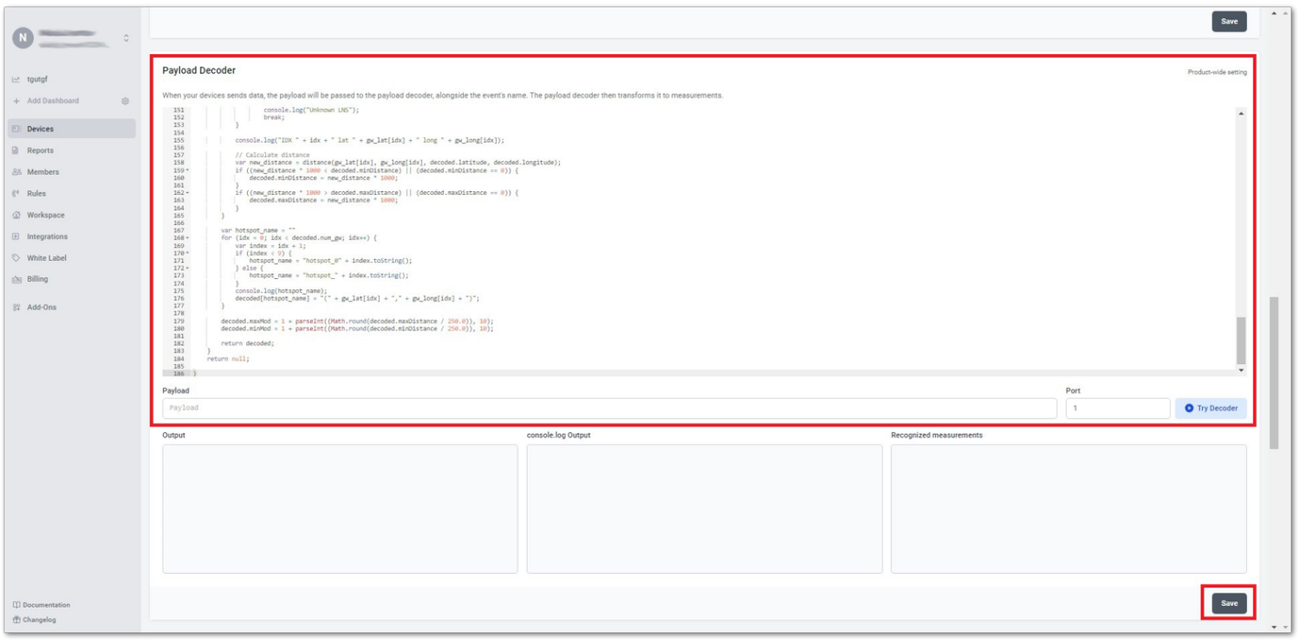

Figure 88: Datacake Payload Decoder Field

2. After saving the payload decoder it is time to set the downlink payload encoder. Navigate to **Downlinks** and copy-paste the provided encoder in the field. Set the port to 2 and use the **IS_CHIRPSTACK** field to trigger the downlink.

```js
function Encoder(measurements, port) {
    var buf = [];
    buf[0] = 1;
    buf[1] = measurements.MINRSSI.value + 200;
    buf[2] = measurements.MAXRSSI.value + 200;
    // var temp = parseInt(measurements.MINMOD.value,10);
    if (measurements.MINMOD.value == 0) {
        measurements.MINMOD.value = 1;
    }
    console.log(measurements.MINMOD.value);
    buf[3] = measurements.MINMOD.value;
    if (measurements.MAXMOD.value == 0) {
        measurements.MAXMOD.value = 1;
    }
    buf[4] = measurements.MAXMOD.value;
    buf[5] = measurements.NUM_GW.value;
    return buf;
}
```

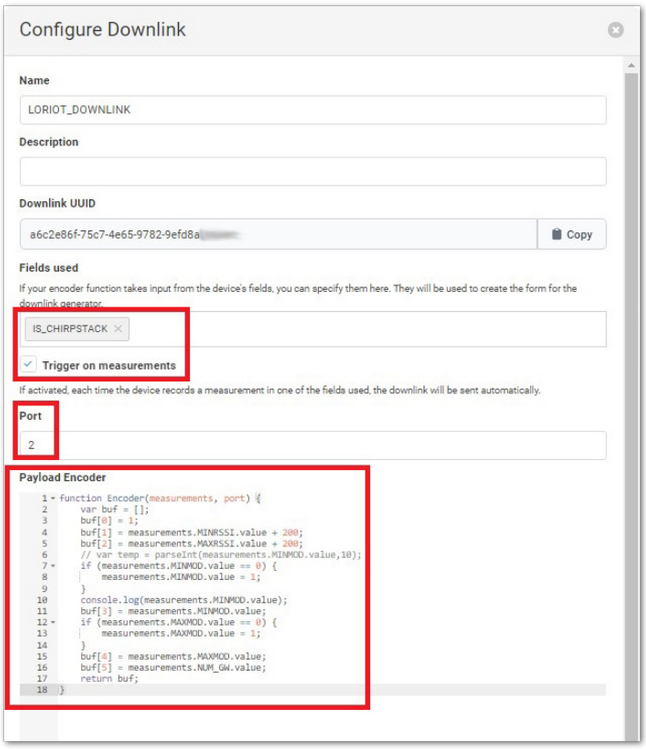

Figure 89: Downlink Configuration

3. You should now be able to see the data from the downlink on your Field Tester's screen.

Configuration of RAK10701-P Using WisToolBox

The Field Mapper should have the correct credentials to connect to the Helium Console. This can be done using WisToolBox and also with the help of the touchscreen LCD user interface.

1. Connect the RAK10701-P to the PC via USB Type-C cable and open the WisToolBox application. You can find more info on how to install and use the WisToolbox from its documentation .

2. Click the **CONNECT DEVICE** button to launch WisToolBox Dashboard.

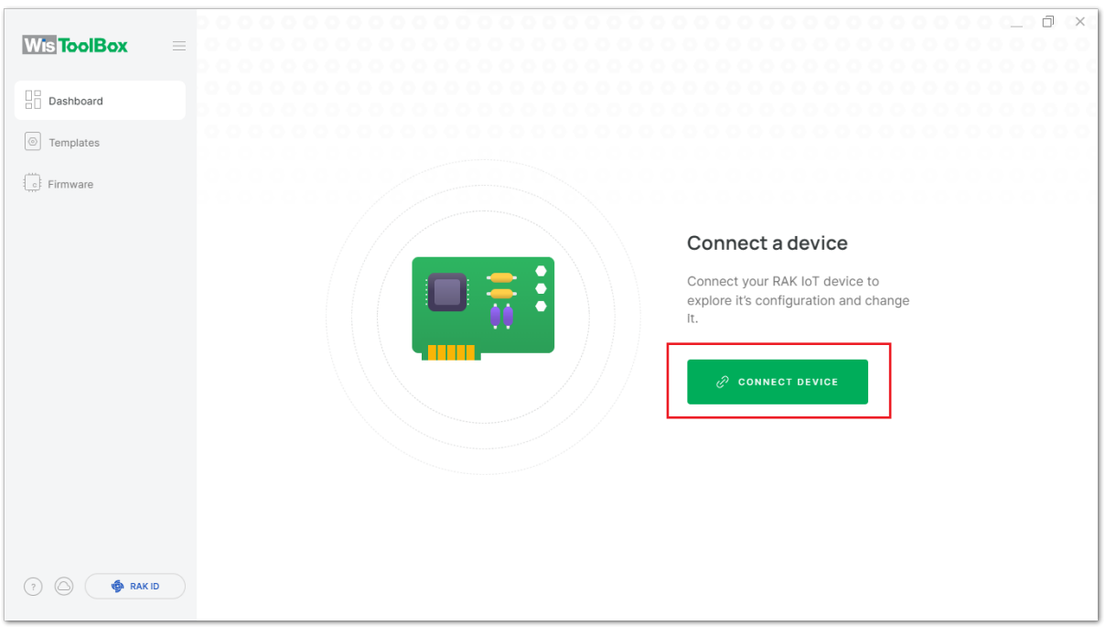

Figure 90: WisToolBox Desktop splash screen

3. Review the **Connection settings** parameters on the dashboard, then click on the **CONNECT** button.

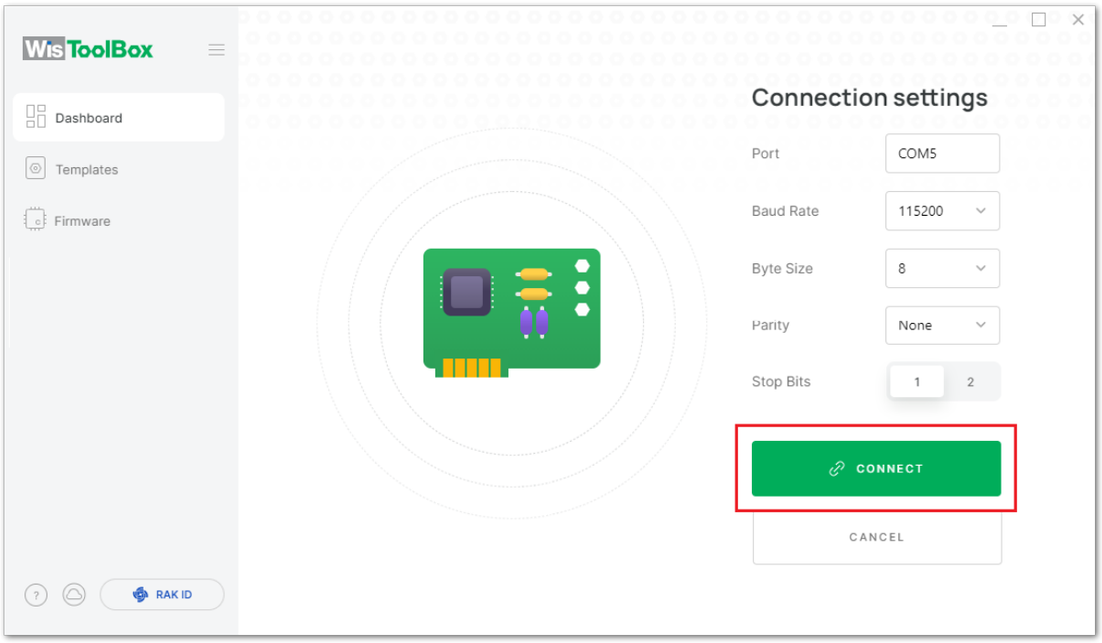

Figure 91: WisToolBox Desktop connection settings

4. On the WisToolBox Dashboard screen, select the RAK4630 module. This is the module inside the RAK10701-P Field Tester device.

Figure 92: Detected module by WisToolBox

5. You can now update the APPEUI, DEVEUI, and APPKEY. It must be the same as the parameters you have in the Setting up of Network Server. Then you can click **APPLY COMMANDS**.

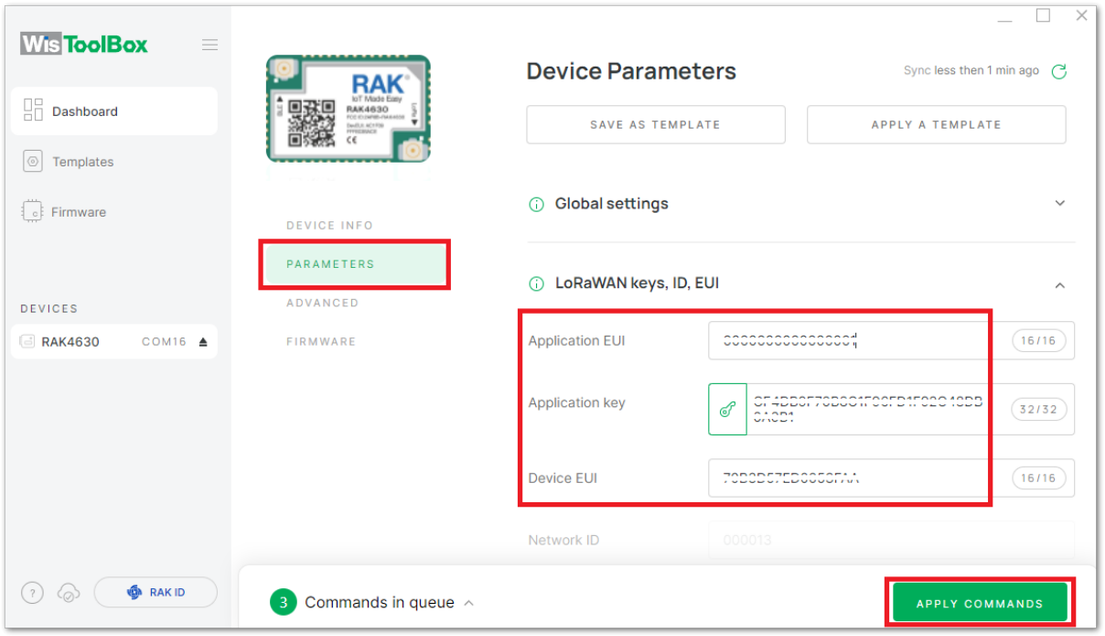

Figure 93: Changing device parameters

> 📝 **NOTE:**
>
> These are the only parameters that you need to change via WisToolBox. Other configurations like frequency plan, the interval of uplinks, TX power, and data rate can be done on the touchscreen of RAK10701.
>
> For the frequency plan change, the device has to restart to activate this newly configured frequency band. There will be a notification on the UI touchscreen LCD. If you use WisToolBox to configure the band, you have to restart the device manually and there will be no notification from the UI of the LCD.

6. You will see the summary of commands that was applied successfully. If the update is unsuccessful, just resend the needed changes. After the successful update, click the **CLOSE** button to return to Dashboard.

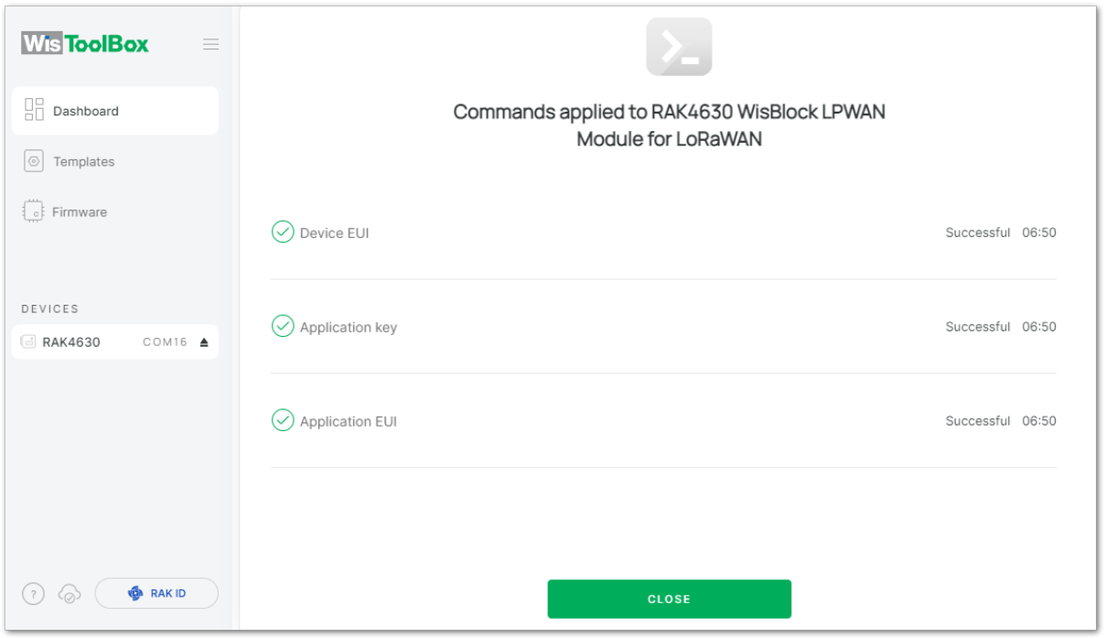

Figure 94: Successful update of parameters

7. You can now remove the USB Type-C cable and proceed to the screen of RAK10701. You can click the settings icon and then update the frequency plan, the interval of uplinks, TX power, and data rate as needed. You can use the arrows for navigation and click `OK` to save changes.

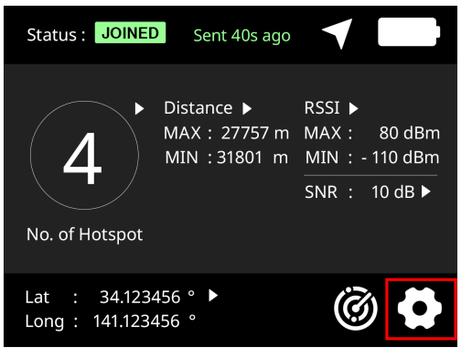

Figure 95: RAK10701-P settings button

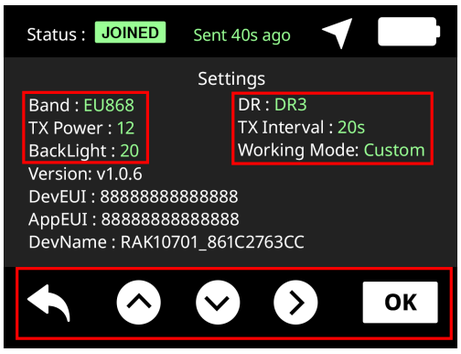

Figure 96: Configurable parameters

Miscellaneous

This part of the guide shows the Field Tester interface and how to update the firmware.

- User Interface
- Firmware Update

Field Tester Display Interface

This section discusses the interfaces on the LCD of the device as well as its pages.

Display Status and Indicator

The RAK10701-P WisNode Field Tester has status indicators that show the current state of the device.

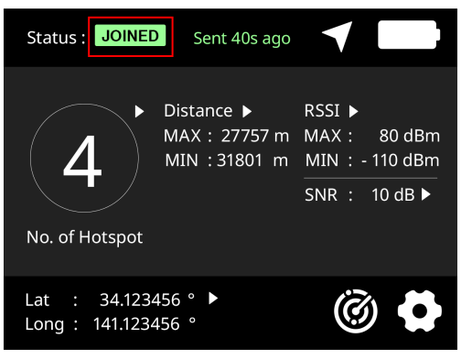

Figure 97: Device status and indicator

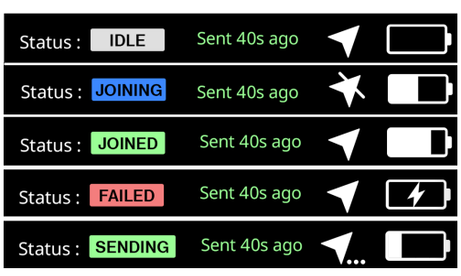

Figure 98: Different device status

Status:

- **IDLE**: RAK10701-P Field Tester Pro state is in between the previous uplink and the next uplink. The duration of IDLE depends on the interval configured on the device.
- **JOINING**: RAK10701-P Field Tester Pro is trying to join the network. This status is triggered when a Join request is sent.
- **JOINED**: RAK10701-P Field Tester Pro successfully received the Join accept the packet. This status will be displayed until refreshed when new data is sent.
- **FAILED**: RAK10701-P Field Tester Pro failed to join the network. Triggered by receive timeout. There might be no available gateway reachable by the Field Tester.
- **SENDING**: RAK10701-P Field Tester Pro's data such as GPS is being uploaded via an uplink. It will be displayed until the reception is completed or timed out.

Settings

The field tester has configurable parameters: Band, TX power, TX interval, backlight intensity, and DR. You can navigate the settings using the arrow widgets plus the back and ok buttons. The OTAA parameters APPEUI, DEVEUI, and APPKEY are also displayed but can't be changed on the touchscreen. WisToolBox or another Serial Port terminal tool is needed to send the AT commands to update the EUIs and key.

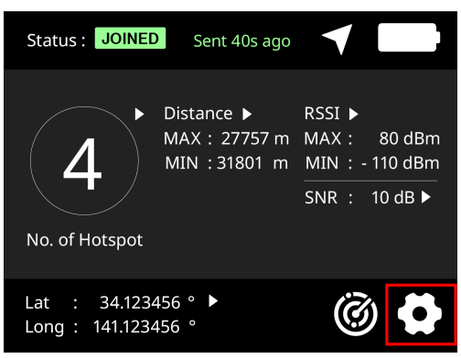

Figure 99: Settings button

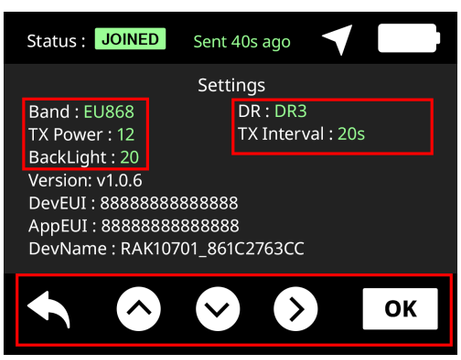

Figure 100: Settings page

Data Plots

There are four different data plots on the field tester: **number of gateways**, **RSSI**, **SNR**, and **approximate distance**. These graphs are accessible by touching the respective icons assigned to the parameter.

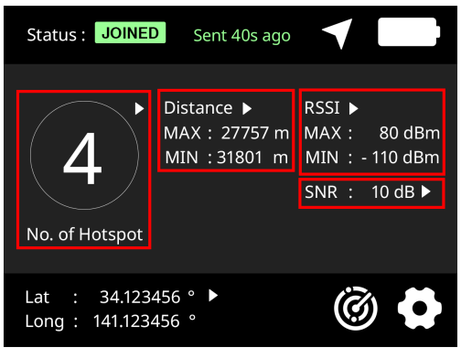

Figure 101: Accessing different data plots

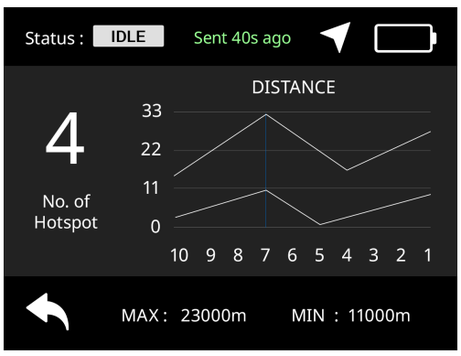

Figure 102: Distance plot

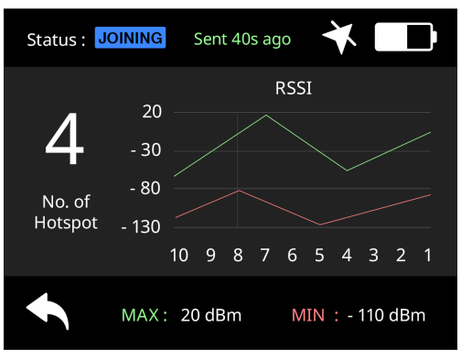

Figure 103: RSSI plot

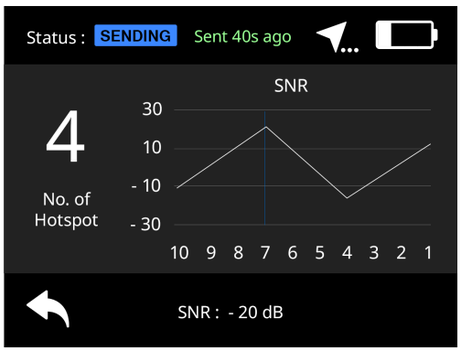

Figure 104: SNR plot

GPS Data

The main page shows the last GPS data captured by the device.

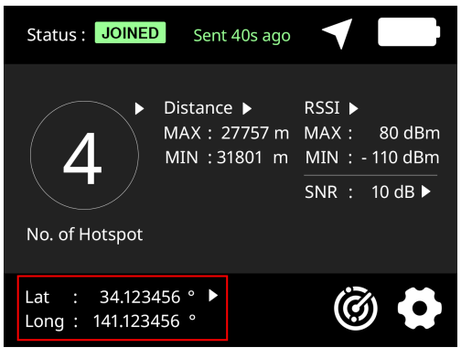

Figure 105: GPS display

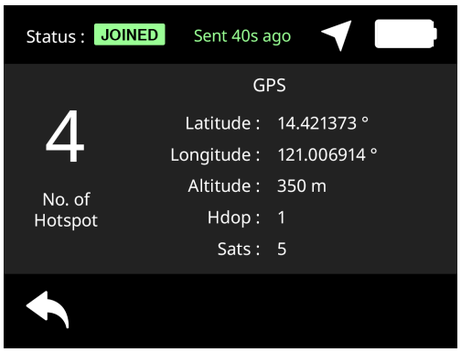

Figure 106: GPS data

Packet Frame Format

The Uplink packet format send on Fport 1:

Byte	Usage
0 - 5	GSP position see here for details. Decoding see below
6 - 7	Altitude in meters + 1000 m (1100 = 100 m)
8	HDOP * 10 (11 = 1.1)
9	Sats in view

When the GPS position is invalid of GPS is disable, the frame is fill with 0's.

The downlink response format send on Fport 2:

Byte	Usage
0	Sequence ID % 255
1	Min Rssi + 200 (160 = -40 dBm)
2	Max Rssi + 200 (160 = -40 dBm)
3	Min Distance step 250 m
4	Max Distance step 250 m
5	Seen hotspot

The distance is calculated from the GPS position and the gateways position returned by LoRaWAN server meta-data. Under 250 m value is 250 m, over 32 km value is 32 km. 0 is considered as invalid response.

The following integration and payload transformation allows to decode the gps position and report is to mapper.

Dicovery uplink format send on Fport 3 (no ack):

Byte	Usage
0 - 5	GPS position

Discovery is sending 10 messages SF10 on every 40 seconds. All the other information comes from the metadata provided by the network server.

Upgrading the Firmware

It is recommended to update to the latest version of the firmware. To do this, download the latest RAK10701-P WisNode Field Tester firmware and use the WisToolBox to update the custom firmware.

1. Drag the downloaded firmware to the WisToolBox custom firmware section.

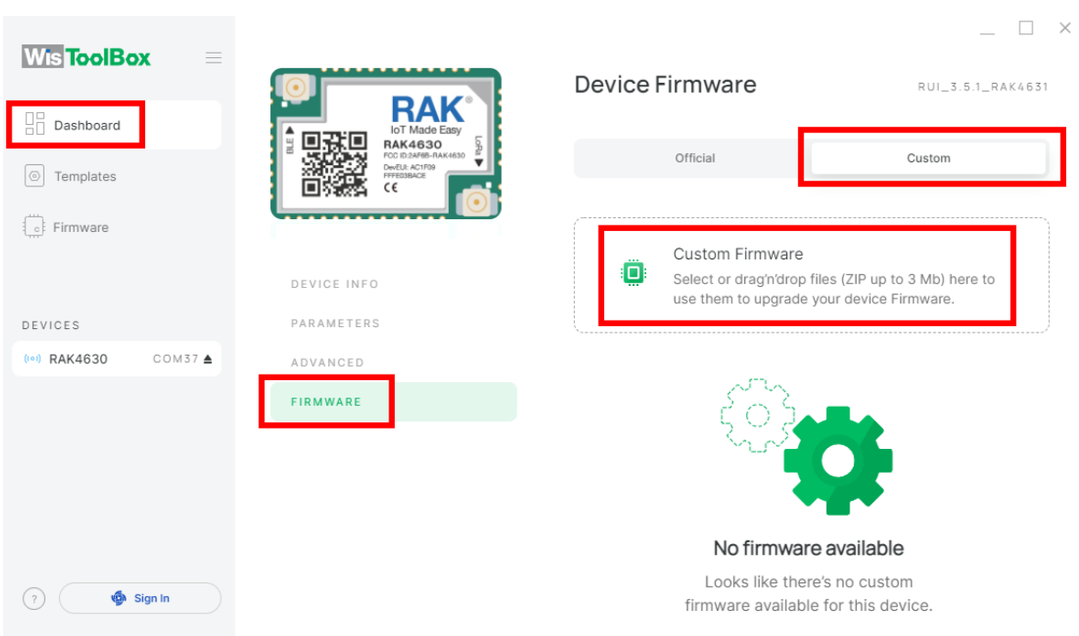

Figure 107: WisToolBox firmware

2. After the firmware file is uploaded to the application, you can now select **UPGRADE DEVICE**.

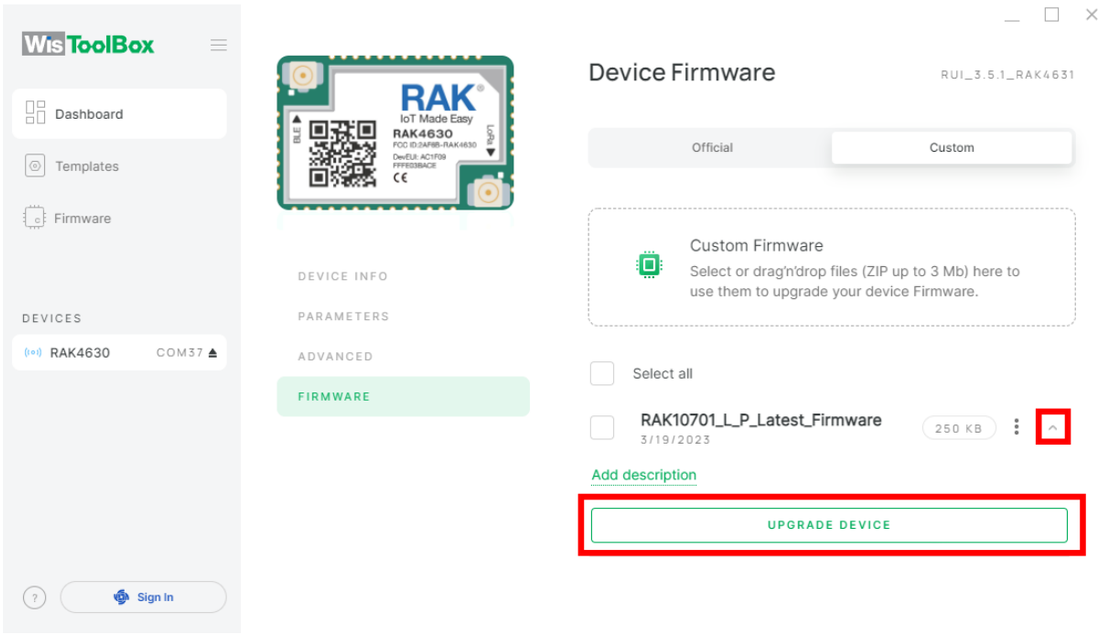

Figure 108: Upload the latest firmware

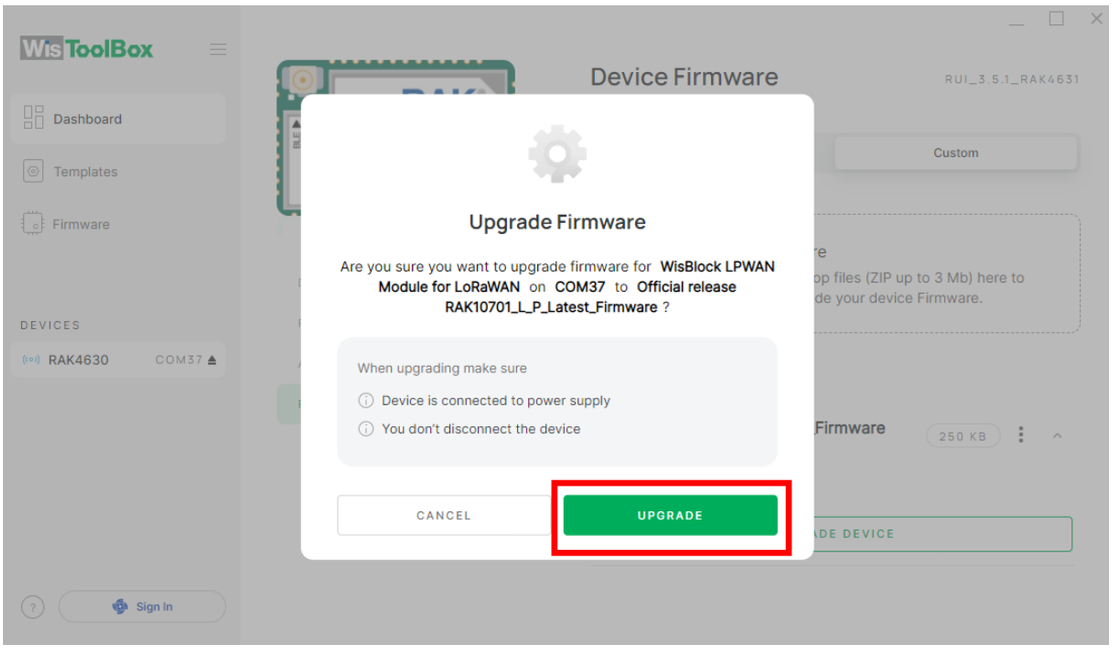

Figure 109: Confirm upgrading of firmware

3. If all proceed with no error, you should see `Firmware update successful` notification, and the RAK10701-P will restart automatically.

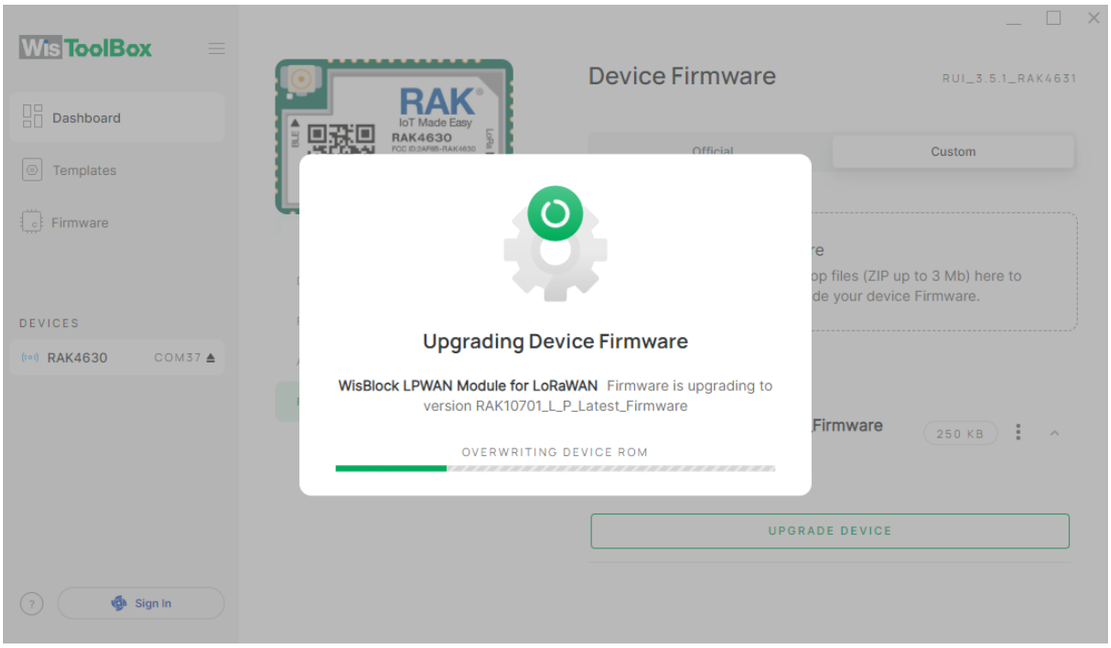

Figure 110: Ongoing upgrading of firmware

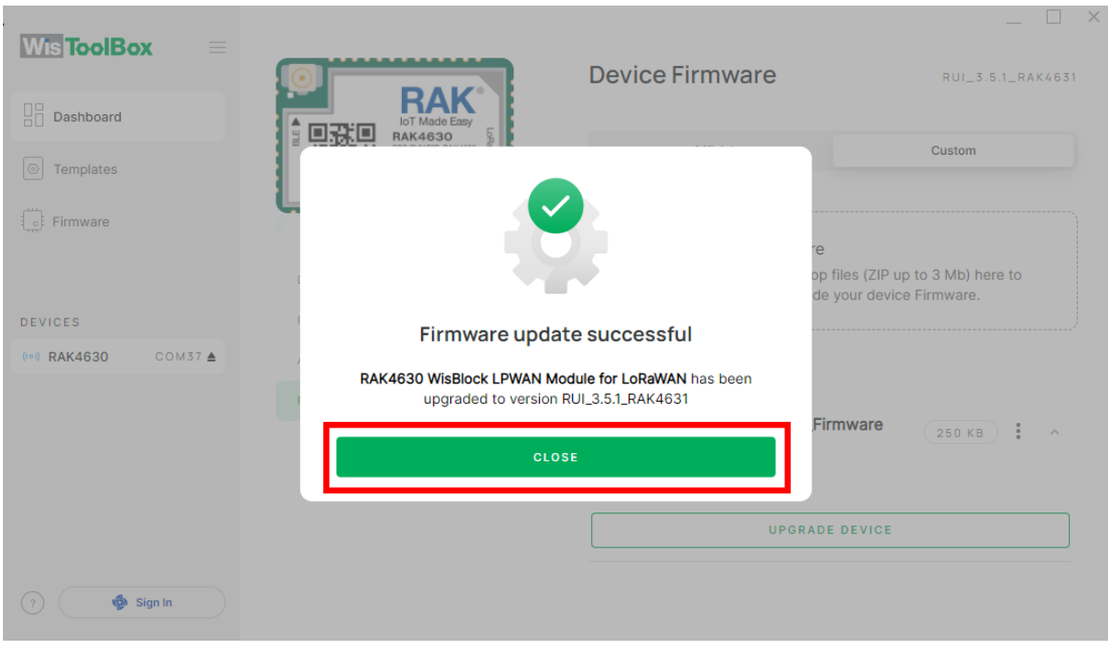

Figure 111: Successful upload of latest firmware

Last Updated: 2/20/2024, 6:23:14 AM