



ADS-260

How to build a Sigfox IoT telemetry application

Vers. 1.4 – Nov 2020

1. Introduction

This document is dedicated to the **ADS-260**, a Sigfox end node that is designed for IoT telemetry applications. It is a battery powered device that can also power sensors. The unit incorporates a Radiocrafts RC1682-SIG (EU/CE) or RC1692HP-SIG (USA FCC /AU/NZ/LATIN AMERICA) Sigfox module that features a unique device ID and PAC code.

In order to realize a Sigfox telemetry system in the context of this document there are some prerequisites,

- PC or laptop with MS Windows
- Internet access
- Recent web browser (Edge, Chrome, Firefox, Safari)
- A valid Sigfox account to the Sigfox Backend and a registered device to the backend.

A valid Sigfox account to the Sigfox Backend and a registered device to the backend is needed. The Sigfox backend is used to manage transmitted data and to configure means to relay data to a front end.

This document describes in brief how to,

- use the Sigfox backend system
- connect devices to Infinite's cloud platform the WaT (web aided telemetry)
- use Losant as a front end system

2. Sigfox Backend

The devices must be registered to the Sigfox backend in order to communicate through the Sigfox Network. The Sigfox backend, allows users to manage devices and view the data they transmit.

In brief we show below as a reference the basic steps that must be taken in the Sigfox Backend.

After having an account created log in to the backend.

<https://backend.sigfox.com/>

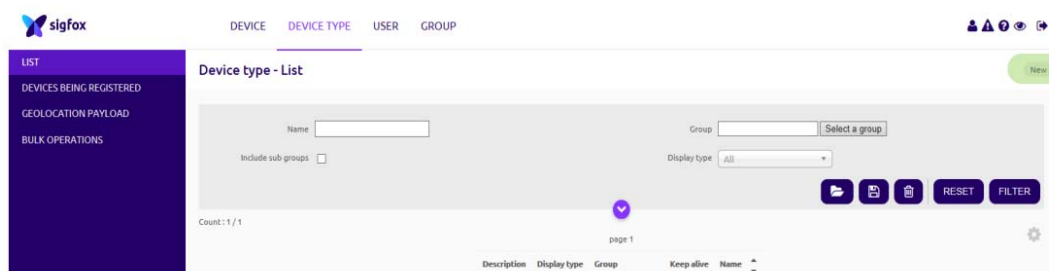
If you have already taken the steps below please omit them and continue with the next ones.

Create Device Type

A Device type is a group of devices. It allows to gather devices and define common means to process the data they transmit.

Device type notion:

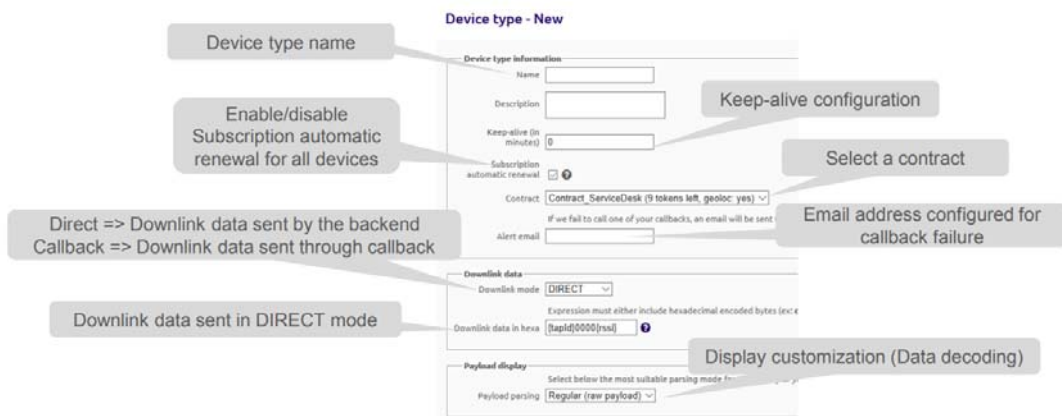
- Set of devices with the same behavior
- Linked to a single order (same subscription levels and duration)
- Belongs to a unique group
- Callback availability to retrieve messages



1. Click on New button in Device type tab.
2. Select a group.



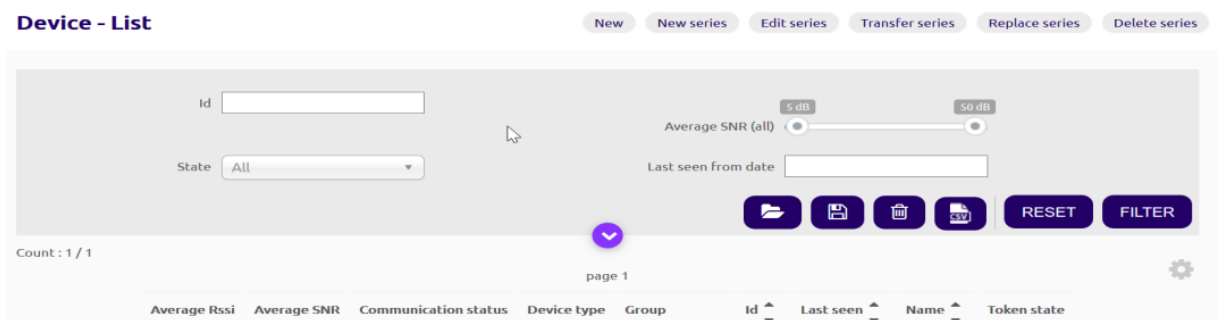
3. Enter device type information.



Manage Devices

A device has a unique Identifier called Device ID. It is also related to another unique number called PAC (Porting Authorization Code). The PAC proves the ownership of a device (ownership title) and only the current device owner knows it.

The ID-PAC couple is mandatory for device registration or transfer. As soon as the device is registered or transferred, the PAC will change (one-time only code).



Select a way to register/manage devices

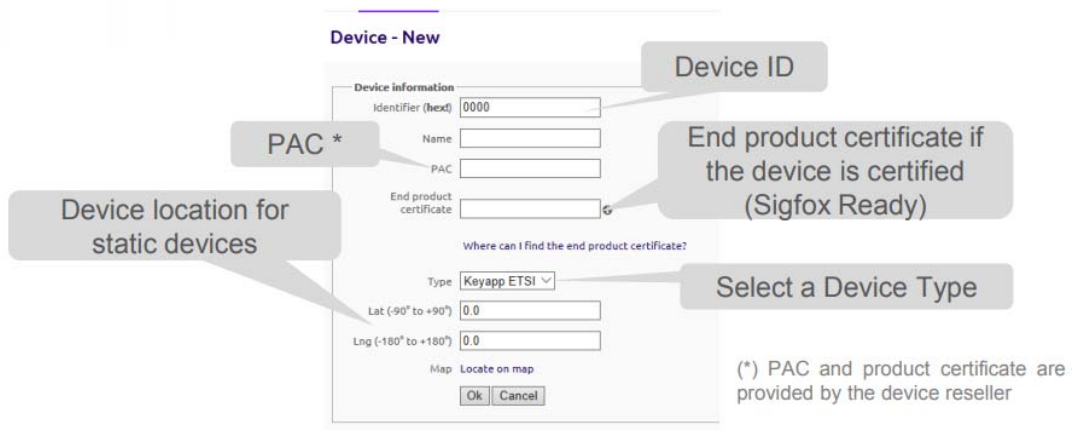
1. New : register devices one by one
2. New series : register batch of devices
3. Edit series : edit device information
4. Transfer series : move devices from device types (same or different contract)
5. Replace series : replace a broken device by a new
6. Delete series : batch of devices deletion

1. If New has been chosen

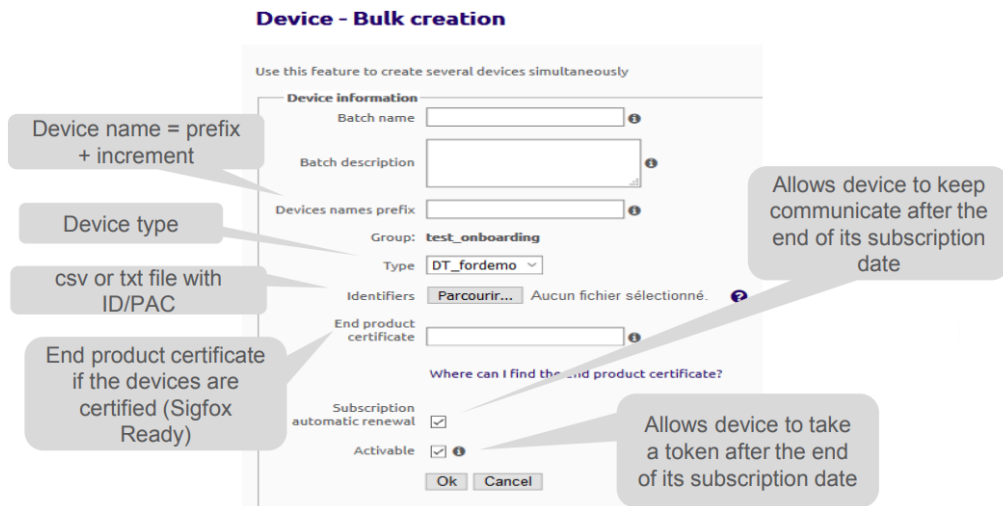
- Select a group to register the device.



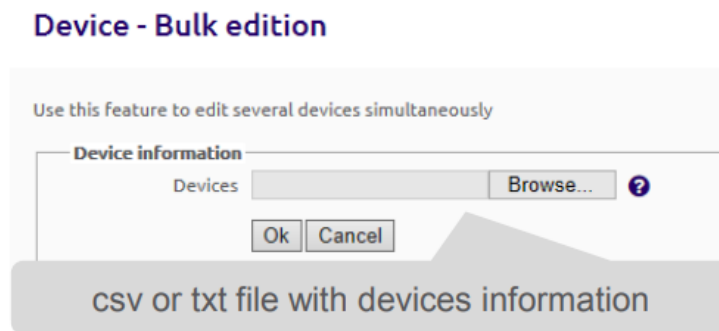
- Enter device information.



2. If New series has been chosen



3. If Edit series has been chosen.

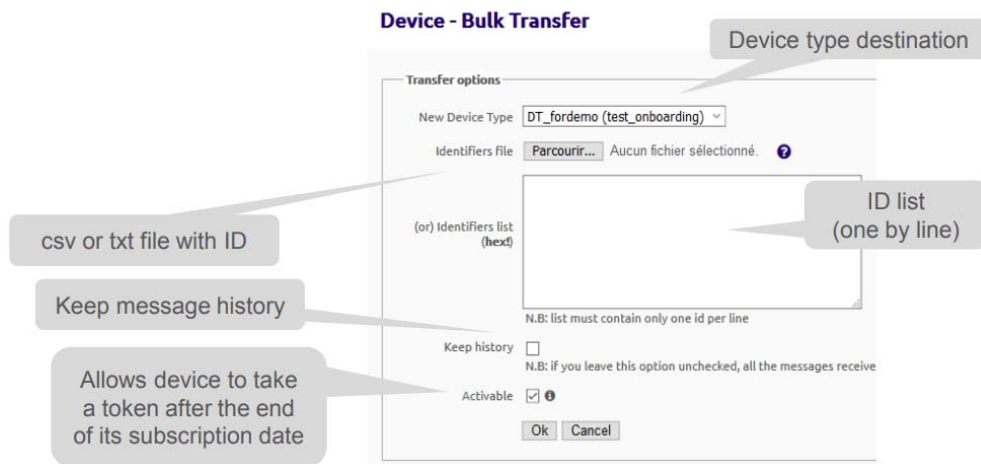


4. If Transfer series has been chosen

- Select the destination group.

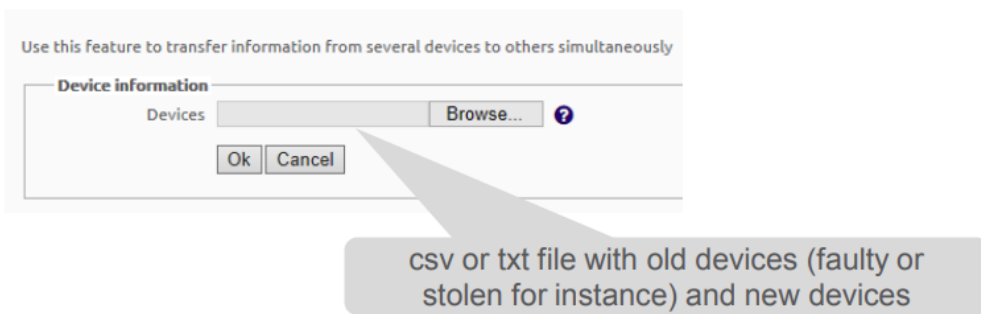


- Enter transfer information.

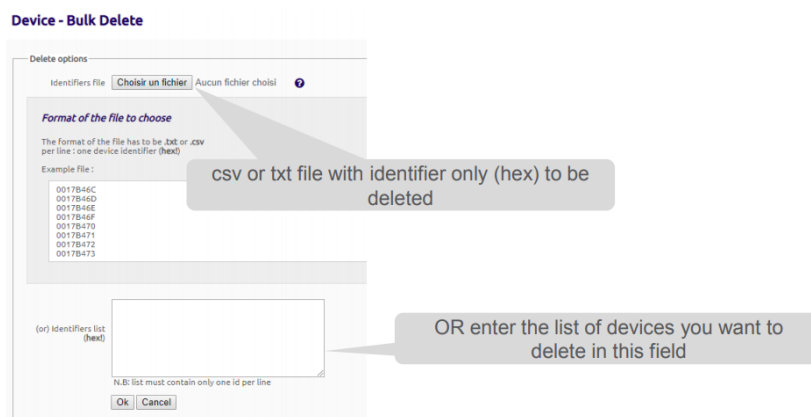


5. If Replace series has been chosen

Device - Bulk replacement



6. If Delete series has been chosen



Sigfox Cloud provides 2 main categories of Callbacks services:

- Data and Service Callback services triggered by a network event (device message or device communication behavior). Those callback services are configured at device type level.
- Event Callback service triggered by an event on the device's entity (its lifecycle, communication status, and subscription) and configured at group level

The available Callbacks services are listed below. All Callbacks services are optional, so subscribe to those that are most relevant for the experience you are trying to create. Multiple callbacks for the same service is allowed, but Sigfox might restrict this possibility in case of abuse.

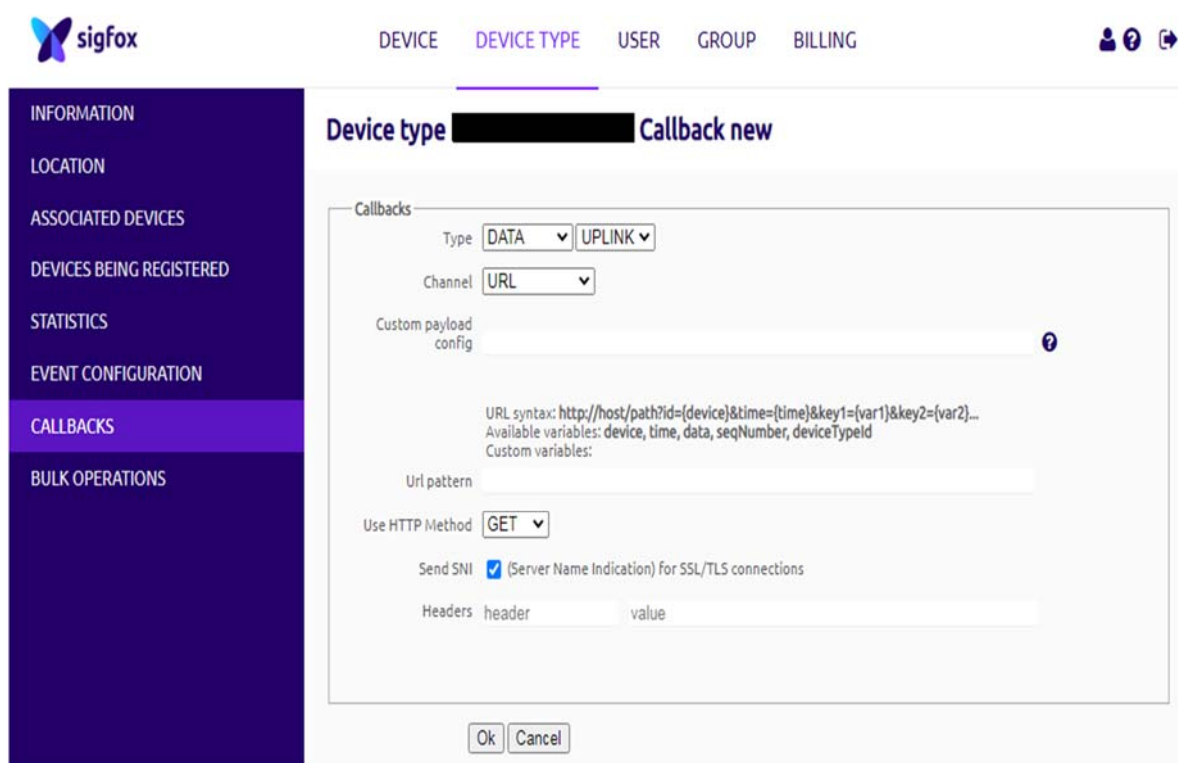
Callback service		Description	Trigger
DATA	Uplink	Send Uplink Message Received event	Reception of the first frame
	BIDIR	Send uplink Message Received and return downlink message	
SERVICE	STATUS	Send device battery and temperature information received for modules allowing Out-Of-Band status messages.	
	ACKNOWLEDGE	Send the network acknowledgment confirming the downlink message emission.	
	REPEATER	service messages (battery, number of repeated messages,...) only applicable for repeater devices.	
	DATA ADVANCED	Send Uplink Message Received with enriched data: <ul style="list-style-type: none"> • Connectivity Metadata (Link Quality Indicator and country of sending) • Geolocation Metadata (Latitude, Longitude, radius, and source - Contract option required) • Network Metadata (Base station Id, RSSI per duplicates - contract option required) 	25 seconds after Reception of the first frame
	ERROR	Send an alert when a device does not comply with the keepalive delay set at device type level.	DeviceType Setting
EVENT	Device events	Send alerts that are triggered upon event occurrence	Event base

You can configure a callback service to forward events directly to your servers using either Custom callbacks, or callbacks for one of the 4 platforms integrated with Sigfox's backend.

Custom callback creation

The configuration of callbacks is done in the device type page for Data, Service, and Error callback service. With following procedure

1. Click on the Device type tab.
2. Click on the name of the device type that you want to create your callback for.
3. Click on Callbacks on the left-hand side menu.
4. Click on the New button, located on the upper-right part of the screen.
5. Choose Custom Callback.
6. Set your Custom Callback.



The screenshot shows the Sigfox web interface. At the top, there is a navigation bar with the Sigfox logo and tabs for 'DEVICE', 'DEVICE TYPE', 'USER', 'GROUP', and 'BILLING'. The 'DEVICE TYPE' tab is active. On the left, a dark blue sidebar menu contains options: 'INFORMATION', 'LOCATION', 'ASSOCIATED DEVICES', 'DEVICES BEING REGISTERED', 'STATISTICS', 'EVENT CONFIGURATION', 'CALLBACKS' (highlighted), and 'BULK OPERATIONS'. The main content area is titled 'Device type [redacted] Callback new'. A 'Callbacks' modal form is open, containing the following fields and options:

- Type: |
- Channel:
- Custom payload config:
- URL syntax: `http://host/path?id={device}&time={time}&key1={var1}&key2={var2}...`
Available variables: `device, time, data, seqNumber, deviceTypeId`
Custom variables:
- Url pattern:
- Use HTTP Method:
- Send SNI: (Server Name Indication) for SSL/TLS connections
- Headers:

header	value
<input type="text"/>	<input type="text"/>

At the bottom of the modal are 'Ok' and 'Cancel' buttons.

Callback type

Callbacks can be of three types: Data, Service, and Error.

Callback type		Description
DATA	UPLINK	Send uplink messages to a customer platform.
	BIDIR	Send uplink messages to a customer platform and waits for a downlink message from the same platform.
SERVICE	STATUS	To retrieve device battery and temperature information provided by service messages (e.g. keep-alive messages).
	ACKNOWLEDGE	To retrieve the network acknowledgement, confirming the downlink message transmission.
	DATA_ADVANCED	To retrieve optional data, such as geolocation, as well as metadata information. The list of available variables is displayed on the backend upon creation. Some variables are computed over the different base stations which received the messages. This callback is therefore delayed by approximately 30s.
ERROR		In case of communication failure, it informs on whether it is a device issue (based on the keep-alive value defined in the device edition page) or a network issue
EVENT		To be alerted upon device event occurrence. This can be configured at the Group, Device Type or Device level.

Data callbacks are the most immediate use of Callbacks when approaching Sigfox data retrieval. You can create an uplink or a bidirectional callback:

- UPLINK: Used to deliver uplink messages to a customer platform.
- BIDIR: Same as the UPLINK type and includes a downlink payload processing.

Channel

Choose communication means.

URL: to push the data to a single URL destination.

- BATCH_URL: This option pushes the data transmitted by devices within a 1 second time period.
- EMAIL: to get the callback data in your email address.

Custom payload configuration

This option allows the user to decode the payload in distinct, simple variables within the Backend GUI.

Note that the custom payload has only a few available types and cannot distinguish different frames format types: all frames will be split the same way.

For more information, please click on the question mark icon in the Backend GUI callback creation page.

URL pattern

Define the HTTP request to be pushed to your server, including available variables. An example is displayed in the GUI.

Displaying a specific port is supported for the URL channel, as follows:

- Add variable as path parameter :

```
https://my_server_address.com:port_number/{variable}/path
```

- Add variable as request parameter :

```
https://my_server_address.com:port_number/path?var={variable}
```

HTTP method

Three HTTP methods can be used:

- GET
- POST
- PUT

SSL/TLS configuration

It is highly recommended to enable the Server Name Indication (SNI) to specify the target domain at the beginning of the SSL/TLS handshake process. As this option is transparent if TLS is not used (regular HTTP URLs), you should let this option checked unless very specific cases.

SNI

Note that the SNI option is enabled by default when a callback is created.

Navigate to <https://support.sigfox.com/docs/> for more detailed instructions.

3. Configuring the ADS-260 unit

The ADS-260 needs to be configured with the,

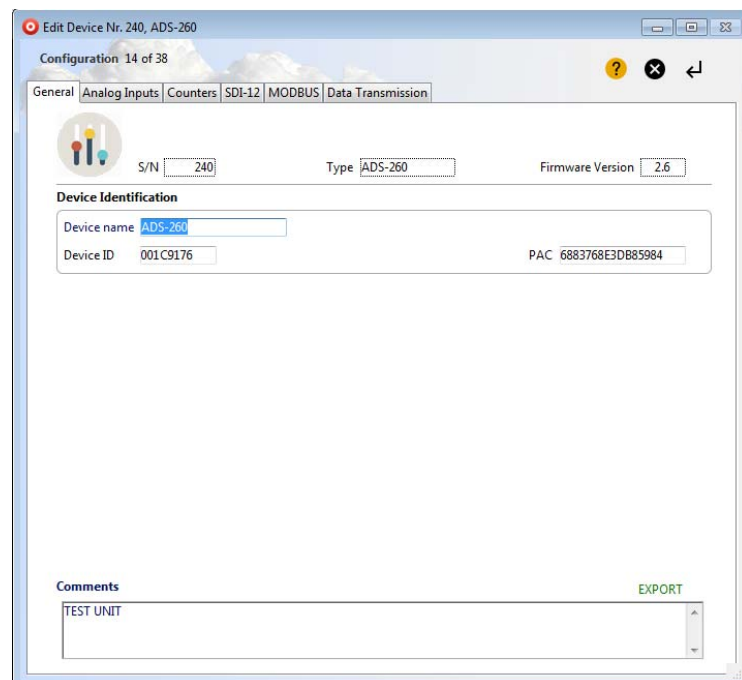
- Unique ID per device
- One property title for each one: PAC (Porting Authorization Code)

The way to enter the unit setup mode is the following:

1. Install the USB driver on a PC.
2. Connect the USB port to a PC. The Status LED is lighting for 2 sec.
3. Put a Jumper on JMP1. The Status LED starts blinking, indicating setup mode. Program execution is suspended.

There are two ways to program the unit:

1. Connecting the unit to a PC and using a terminal program to pass the ASCII commands to the unit, according to the scheme: "Command, Parameters <CR>". The terminal settings should be: Baud rate: 115200 bps, Data bits: 8, Parity: none, Stop bits: 1, Flow control: none.
2. Connecting the unit to a PC and using the WA Manager software. This is the most convenient way. The Device ID, which is necessary for connecting the unit to the Sigfox network, is automatically read during downloading the parameter file to the device. The Device ID is also printed on a label in the device interior.



For sensor configuration, transmission rates, battery life and all the functions of the ADS-260 please consult the device manual.

Removing the Jumper at JMP1 will exit the configuration mode and set the device in operation mode.

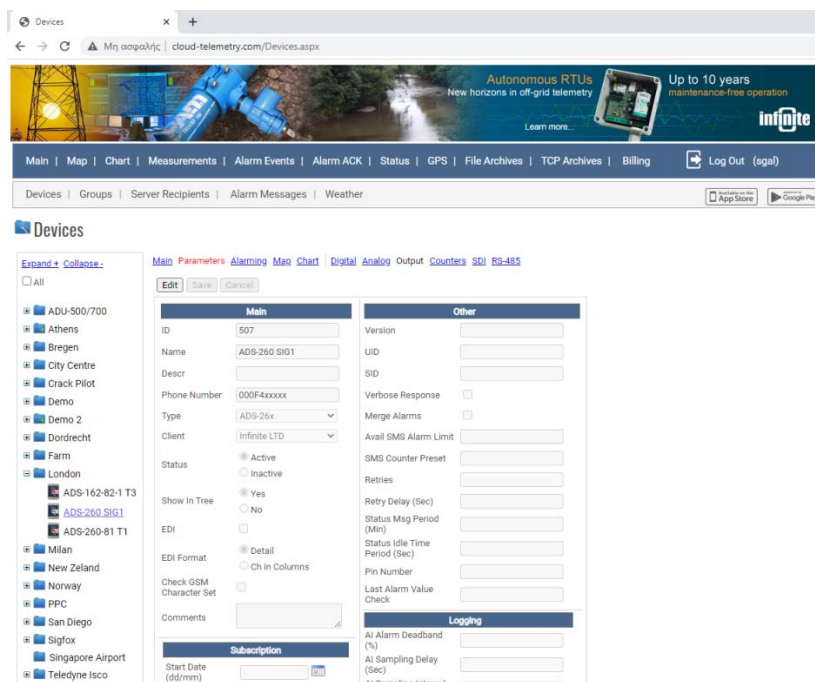
4. Configuring an ADS-260 with Infinite's cloud applications.

Infinite offers a variety of cloud applications to manage and visualise device data.

These include

- The WaT (web aided telemetry platform). Accessible at www.cloud-telemetry.com
- The WaT Eye (live weather data dashboard). Accessible at <http://91.138.204.120:14616/>
- The WaT smart applications for IOs and Android phones and tablets.

The only prerequisite for an ADS-260 to be recognized automatically by the above applications is to configure at the device parameters, at the Descr field, the Device ID.



5. Losant Enterprise IoT platform

An example integration to a 3rd party cloud platform will be given to demonstrate the capabilities of the ADS-260 Sigfox unit. For this demonstration the Losant platform was chosen.

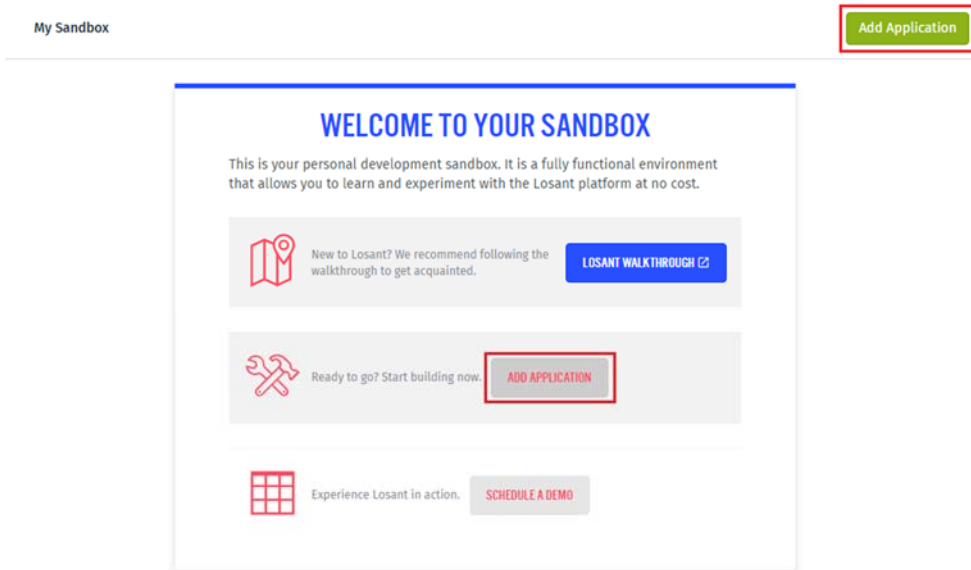
Losant is an easy to use, modern, and powerful Enterprise IoT platform designed to allow rapid build of real time connected solutions.

It is an application enablement platform which allows enterprises to effectively build applications that securely scale to millions of devices. With real-time stream processing and batch processing capabilities, users can create dynamic experiences and perform complex analytics.

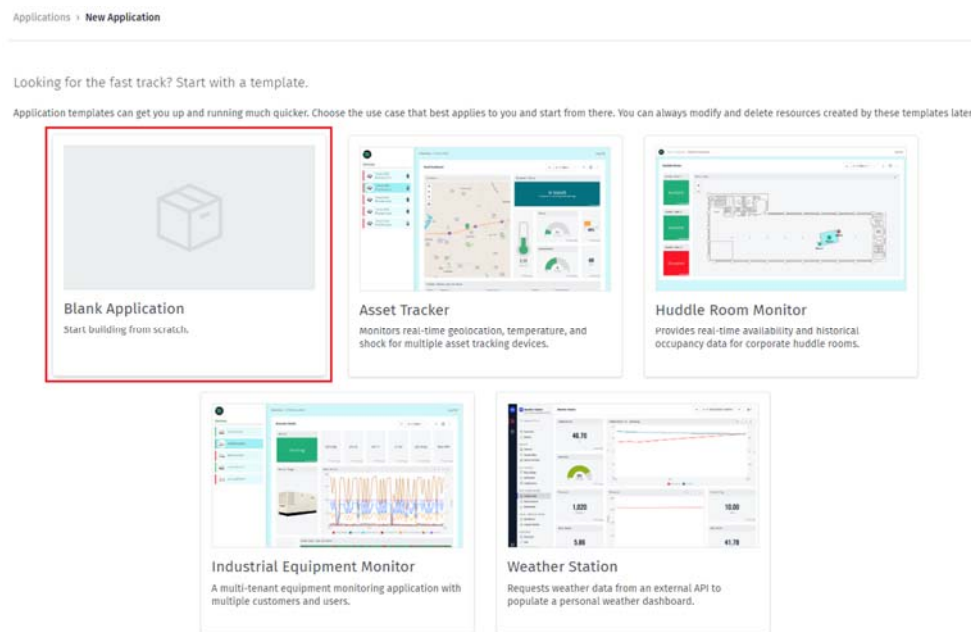
5.1 Create Application

A free user account with Losant is required. An account can be created at www.losant.com

After creating an account, the user will be prompted to create an application. Select "Add Application" to create a new application.



Select the "Blank Application" template.



Provide a name and an optional description for the application and then select "Create Application".

Blank Application

Application Name
e.g. My Great Application

Description
e.g. My new application description

Create Application Cancel

5.2 Create Webhook for Sigfox Callback

The easiest way to communicate between Sigfox and other services is to use a callback. Callbacks allow Sigfox to send any information reported by a device to another service using a webhook.

Select "Webhook" from the Application Menu.

sigfox_test
From My Sandbox

Webhooks > New Webhook

NEW WEBHOOK

After you create your new webhook, you will be assigned a unique URL for making your requests.

Webhook Name
e.g. My Great Webhook

VERIFICATION

Some webhook providers require the endpoint to be verified. Losant will automatically respond to verification requests for the following providers. If you are attempting to use a webhook provider that requires verification and is not listed below, please [let us know](#).

No Verification Alexa Facebook Messenger Fitbit Twilio

Verification Code Template
e.g. {{globals.webhook.verify}}

Response Code
200

BASIC AUTH

You can optionally choose to require basic auth for requests against this webhook.

Basic Auth Username Template
e.g. {{globals.username}}

Basic Auth Password Template
e.g. {{globals.password}}

CUSTOM REPLIES

You can optionally choose to configure this webhook to wait for a reply from a workflow. When checked, this means that when a request is made against this webhook, the request will wait for a Webhook Reply node to be executed in a workflow for the particular request, and that reply will be returned. If no workflow provides a reply within 30 seconds, the request will be timed out.

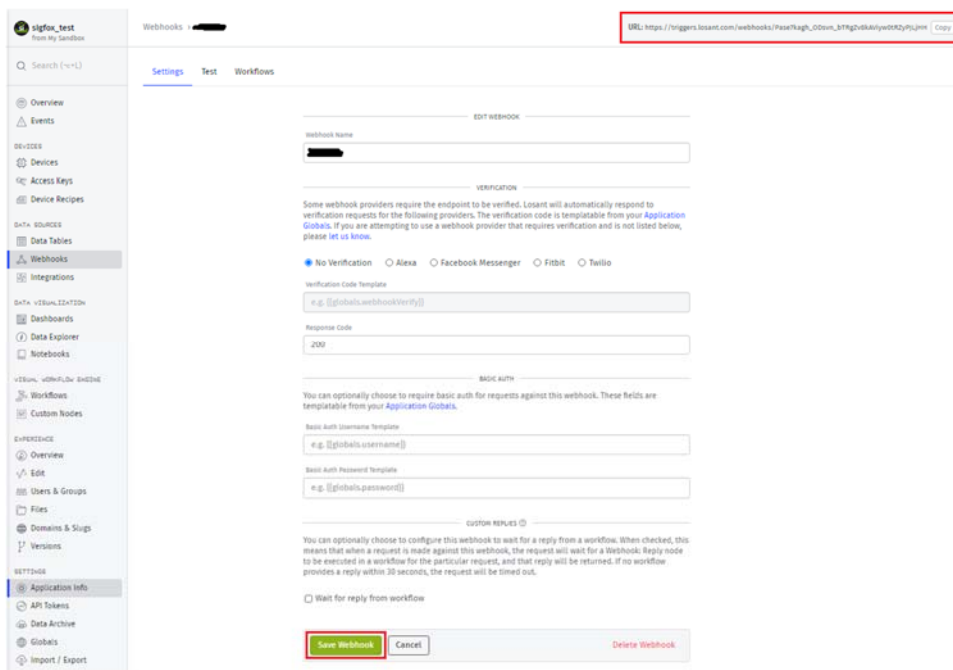
Wait for reply from workflow

Create Webhook Cancel

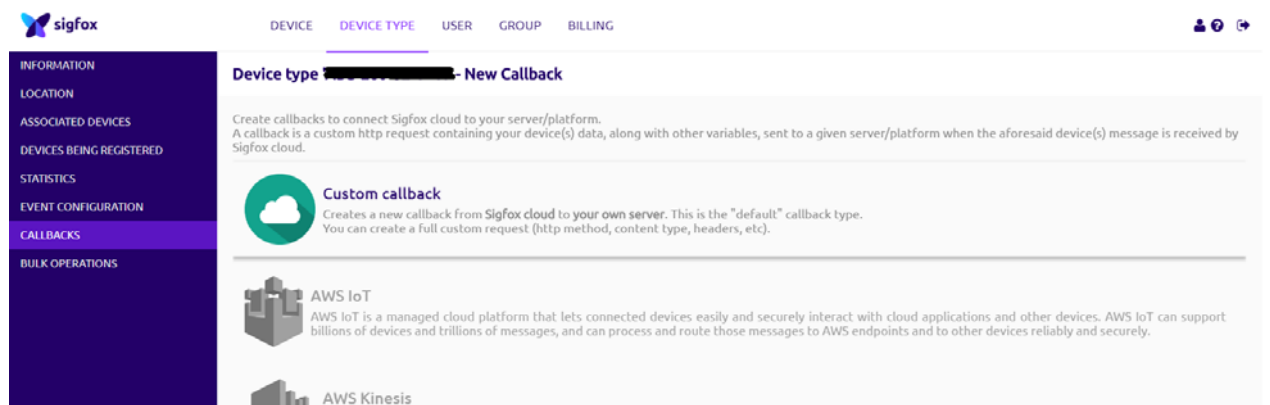
Name the Webhook and press the "Create Webhook" button. . After that a URL will

appear.

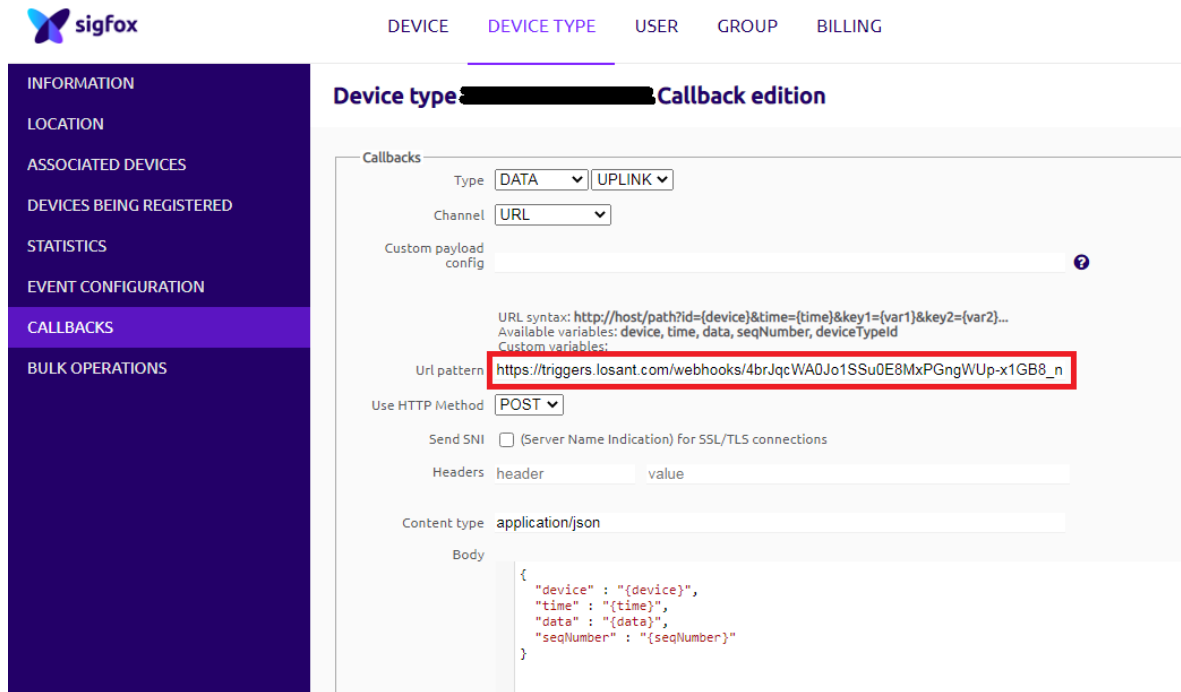
Copy this URL.



Navigate to Sigfox backend and select "Callbacks" from the Device Type Menu. Click on "New" button and select "Custom Callback".



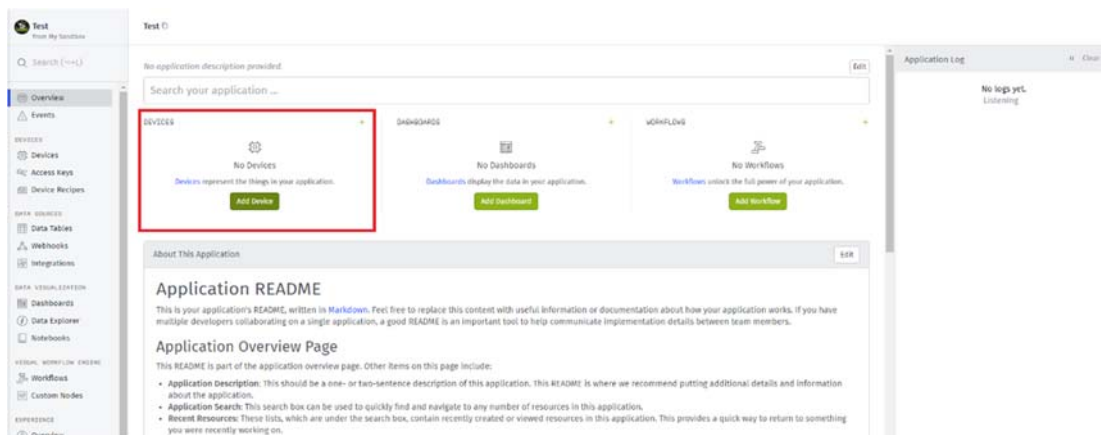
Fill the "Url pattern" gap with the URL that you copied before and adjust all the settings as shown in the image below.



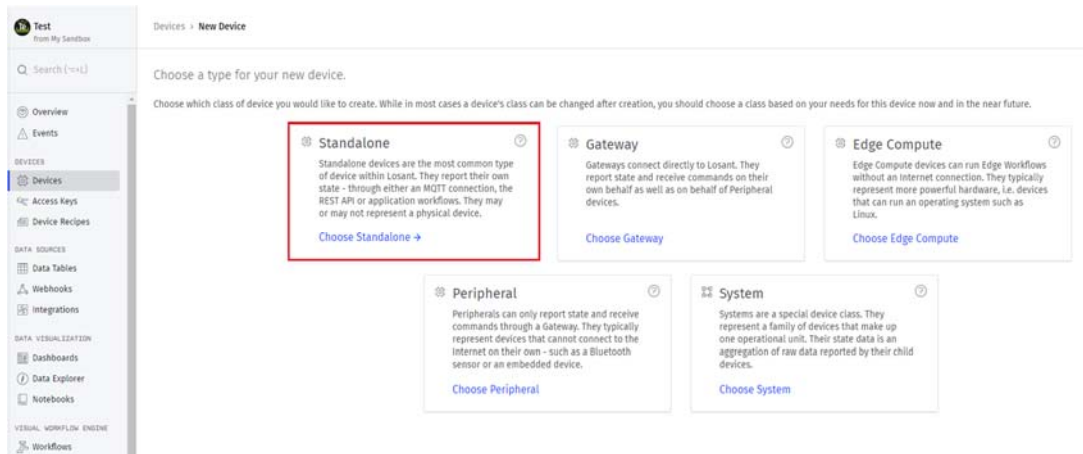
After that, click on "Ok" to save the callback.

5.3 Add the Device

Back to Losant, a device must be created to represent your Sigfox device. Select "Add Device" from the Application Overview page.



When creating the device, select "Standalone" as the Device Class.



A device must be configured.

1. Name the device anything you want.
2. Add a tag named "sigfox_ID" that holds the Sigfox ID of your device. This will be used later to look up the appropriate device connect based on incoming data from the Sigfox callback.
3. Add an attribute for each piece of data being pushed by the Sigfox callback.
4. Add custom attributes for each piece of data encoded in your device's data property. This data will be decoded in a later step and will be saved on more usable fields.

DEVICE TAGS

Device tags provide a way to organize your devices. Tags are defined as keys and values. In other parts of the platform, like visualizations, you can query devices by their tags.










Keys may only contain uppercase letters, lowercase letters, numbers, underscores (_) or hyphens (-).

Key	Value	
sigfox_id	F4DDB	-
sigfox_id	F4DE9	-

ATTRIBUTES

The following attributes are currently configured for this device. These attributes can be deleted and optionally recreated, but know that doing so will delete all data associated with the attribute.

Expand All

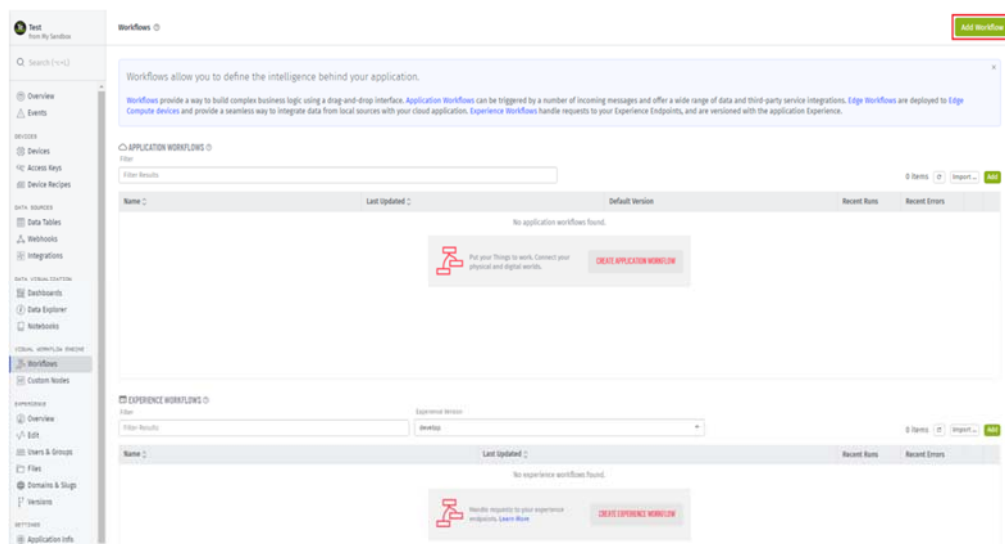
device	
data	
time	
channel	
sign	
seqNumber	
di	
temperature1	
temperature2	

5.4 Workflow

Workflows help describe the logic for applications.

5.4.1 Create a Workflow

To create a workflow, select "Workflows" from the Application Menu. Then, select "Add Workflow."



Name the workflow and press the "Create Workflow" button.

CREATE APPLICATION WORKFLOW

Workflows provide the business logic behind all parts of your IoT solution. This can include processing data, generating notifications, interfacing with your local equipment, or even backing your entire custom user experience.

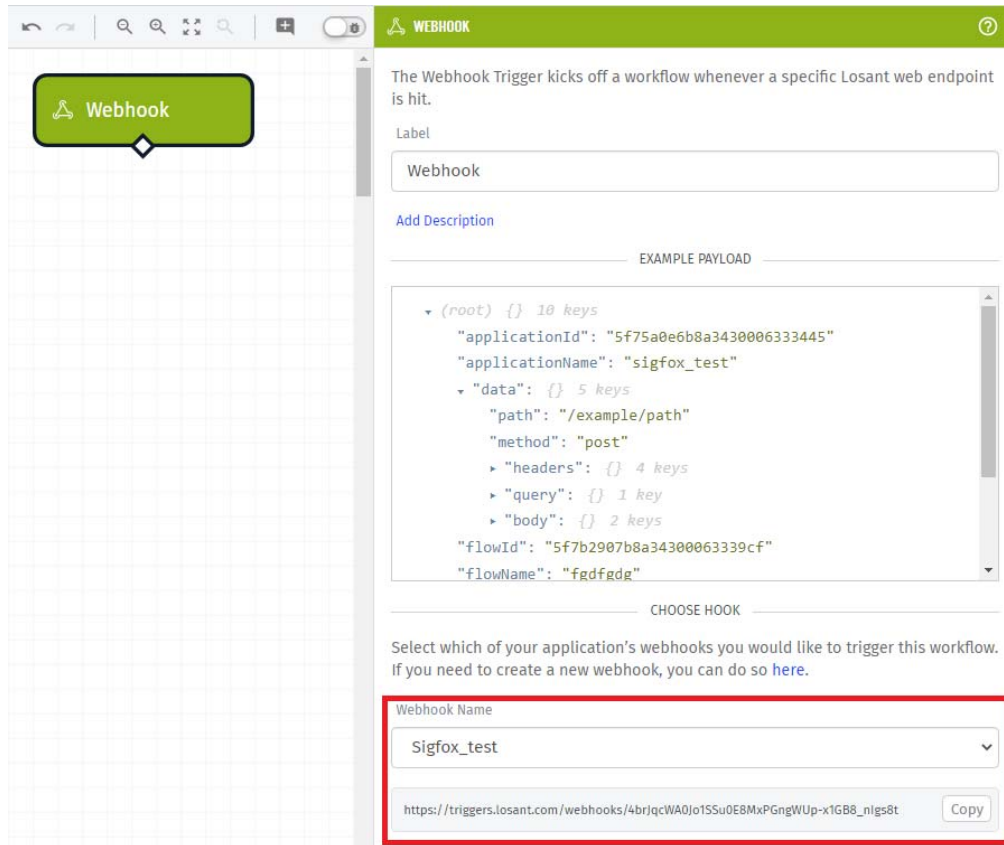
Workflow Name

Description

After creation the workflow canvas is enabled.

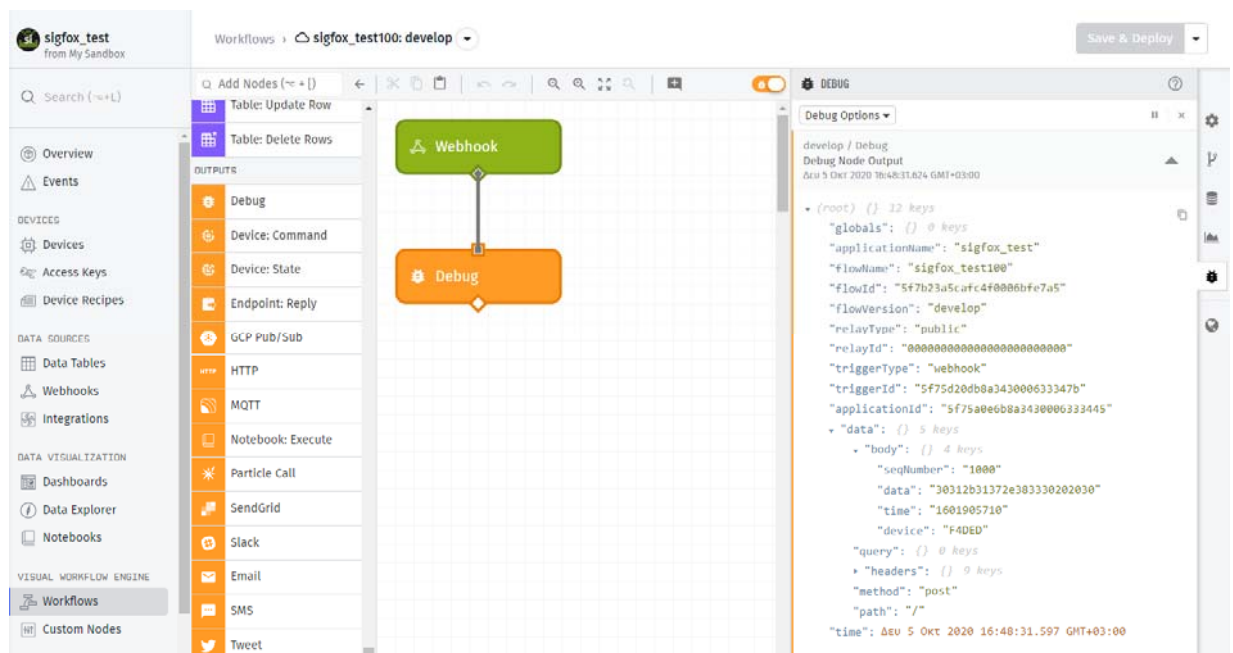
5.4.2 Add a Webhook

Add a Webhook trigger node and a Debug node to verify that you are receiving information from Sigfox.



Select the webhook you created earlier from the dropdown. If it's the only one, it will be selected automatically.

After deploying this workflow, you should see the data that Sigfox is sending show up in the Debug tab (bottom-right corner).



There will be a lot of information on the payload, but what we care about will look something like the following:

```

"data" : {
  "body" : {
    "device" : "...",
    "time" : "...",
    "data" : "...",
    "seqNumber" : "...",
  }
}

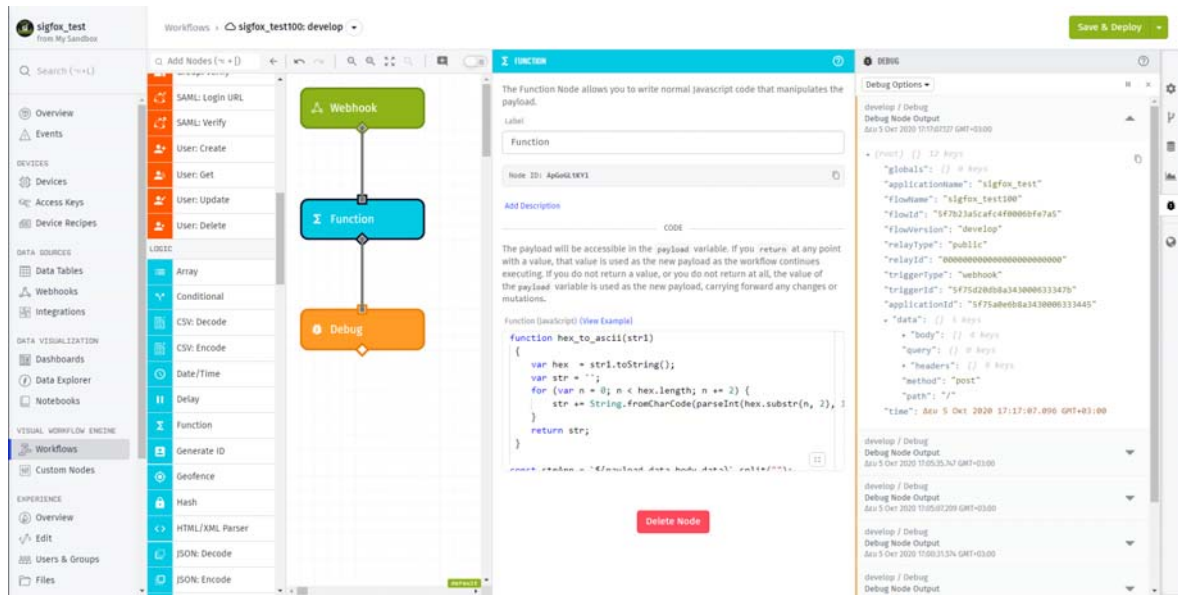
```

This is what Losant calls the "payload". Workflows act on the payload as it flows through nodes. In the payload above, the Sigfox ID of this device is located on the `data.body.device` property.

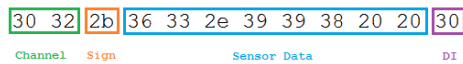
The next thing we need to do is parse your specific data, which is located on the `data.body.data` property. What is in this field is entirely controlled by the users and what the Sigfox device reports. All of the other fields (device, time, seqNumber) are all sent by Sigfox itself.

Since this data could be anything, a Function node is needed to decode it.

5.4.3 Add a Function node



In this example the data string is encoded as shown in the image below.



The code that has been used to decode it is the following.

```
function hex_to_ascii(str1)
{
    var hex = str1.toString();
    var str = '';
    for (var n = 0; n < hex.length; n += 2) {
        str += String.fromCharCode(parseInt(hex.substr(n, 2), 16));
    }
    return str;
}

const strArr = `${payload.data.body.data}`.split("");
var s={};
var i, i_length=12;
var a=[];
var temp={};
for (i=0; i<i_length; i++){
    a[i]=strArr[2*i]+strArr[2*i+1];
}
temp= a[0]+a[1];
s.channel = hex_to_ascii(temp);
s.sign = hex_to_ascii(a[2]);
temp= a[3]+a[4]+a[5]+a[6]+a[7]+a[8]+a[9]+a[10];
```

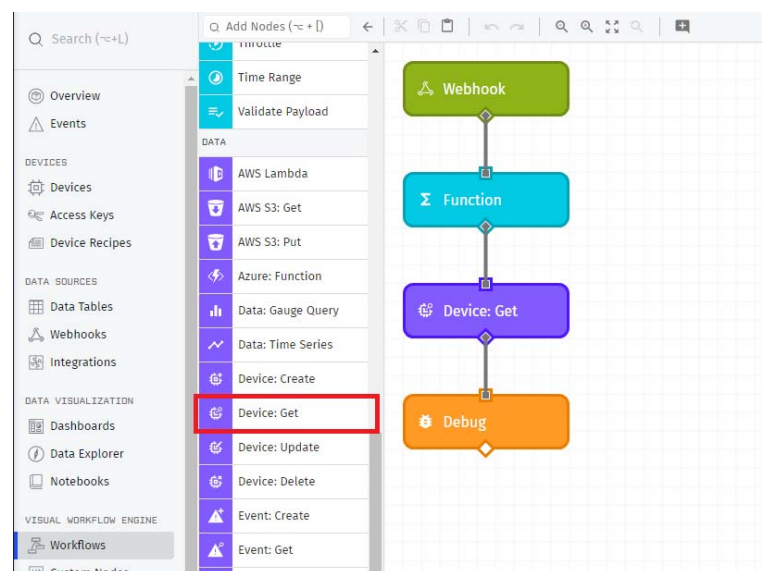
```
temp = hex_to_ascii(temp);
s.value = temp.replace(/\\x20/g, "");
s.di = hex_to_ascii(a[11]);

if (s.sign === "-"){
  s.value *= (-1);
}
if (payload.data.body.device === "F4DDB"){
  payload.data.body.temperature1 = s.value;
}
if (payload.data.body.device === "F4DE9" ) {
  payload.data.body.temperature2 = s.value;
}
payload.data.body.channel = s.channel;
payload.data.body.di = s.di;
payload.data.body.sign = s.sign;
```

This function is decoding the channel, sign, temperature/humidity and the DI and putting the values back on the payload, next to all of the other fields at data.body.channel, data.body.sign, data.body.temperature/humidity and data.body.di.

5.4.4 Add a Get Device node

The next step is to look up the device based on the Sigfox ID, by using the Get Device node.



Select "Device: Get" node from the menu and adjust all the settings as shown below.

🏠 **DEVICE GET** 🔍

The Device: Get Node retrieves one or more devices from your application.

Label

Device: Get

Node ID: FrqINAsmn8 📄

[Add Description](#)

QUERY METHOD

Find one or more devices by choosing a query method and entering search parameters below.

Find by ...

Match Any Tags Query ▼

CONFIGURATION

Any device(s) returned must match any tags. If a key is set without a value, any device that has that key set regardless of the value will be returned. (And vice-versa for values set without keys.) Both keys and values are templatable.

Key Template	=	Value Template	
sigfox_id		{{data.body.device}}	-
Key Template	=	Value Template	

RESULTS CONFIGURATION

You may opt to return multiple devices if more than one device matches your query. You may also configure which results to return from your query by manipulating the per page and sort options. Results per page and results page are templatable.

Sort Field	Sort Direction
Name ▼	Ascending ▼

Return multiple devices?

RESULT

Specify the payload path at which the result of your query will be stored. If returning only a single device, the result will be an object containing the device at the payload path specified below, or `null` if no device is found. If returning multiple devices, the format of the result will depend on if you are including metadata in the result.

If you also want to know the last reported value for attributes on the device, you can request that some or all of the most recent composite state be included on the device (it will be placed on the property `compositeState`).

You may also opt to return the device's tags as an object instead of as the standard array.

Composite State To Include

Include no attributes ▼

Select Attributes

Select at least one ... ▼

Return tags as an object map instead of an array

Result Path

data.deviceResponse

Delete Node

5.4.5 Add a Device State node

The last thing is to use a Device State node to save this data onto the device.



Select "Device State" node from the menu and adjust the settings.

A screenshot of the configuration panel for the 'DEVICE STATE' node in Node-RED. The panel has an orange header with the text 'DEVICE STATE'. Below the header, there is a description: 'The Device: State Node allows you to set the state of any device you've created.' The configuration includes a 'Label' field with the value 'Device: State', a 'Node ID' field with the value 'vsTAWHR13N', and an 'Add Description' button. There are two radio buttons: 'Select a specific device' (unselected) and 'Use a Device ID specified on the current payload' (selected). Below the radio buttons, there is a 'Device ID' dropdown menu with the value 'Select one device ...'. The 'Device ID JSON Path' field contains the value 'data.deviceResponse.id'. There is a 'STATE' section with a 'Data Method' dropdown set to 'Individual Fields'. Below this, there is a list of attributes to be updated, each with an 'Attribute' field and a 'Value' field. The attributes and their values are: 'channel' with value '{{data.body.channel}}', 'data' with value '{{data.body.data}}', 'device' with value '{{data.body.device}}', 'di' with value '{{data.body.di}}', 'seqNumber' with value '{{data.body.seqNumber}}', and 'sign' with value '{{data.body.sign}}'. Each row has a red minus sign to its right.

Attribute	Value
temperature1	{{data.body.temperature1}}
time	{{data.body.time}}
temperature2	{{data.body.temperature2}}
e.g. sensorValue	e.g. {{data.foo}}

TIME

Finally, control the time value that should be associated with this new state data. By default, the time of the current payload is used, but can be changed to either the current time or any time value on the payload.

Use the time of the current payload
 Use the current time
 Use the time at the specified payload path

Time Payload Path

RESULT PATH

Optionally, you may store an object at a path on the payload which indicates the success or failure of queuing this device state change. A success does not guarantee the device state will update.

Payload Path for Result

Delete Node

After all this steps the workflow must look like the following image.

The screenshot shows the Sigfox workflow editor interface. The workflow consists of the following nodes in sequence:

- Webhook (Green)
- Function (Blue)
- Device: Get (Purple)
- Device: State (Orange)
- Debug (Orange)

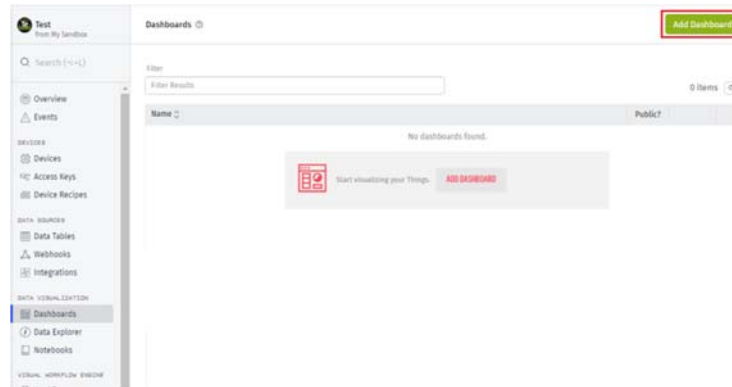
The Debug console on the right shows the output of the Device: State node. A red box highlights the 'data' field in the response:

```

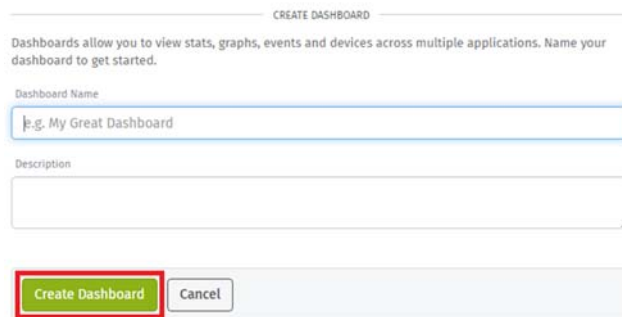
{
  "globals": {},
  "applicationName": "sigfox_greece",
  "flowName": "sigfox_greece",
  "flowId": "5fae59acc076040006c02f2f",
  "flowVersion": "develop",
  "relayType": "public",
  "relayId": "000000000000000000000000",
  "triggerType": "webhook",
  "triggerId": "5fae74ed0ab31b00070c0b2",
  "applicationId": "5fae4d4c2c0b500002f37db",
  "data": {
    "deviceResponse": {
      "body": {
        "sign": "a",
        "di": "-",
        "channel": "01",
        "temperatures": "000010.1",
        "sensor": "1122",
        "data": "505120303030315026312d",
        "time": "1605506033z",
        "device": "FAD00"
      }
    }
  }
}
    
```

5.5 Create a Dashboard

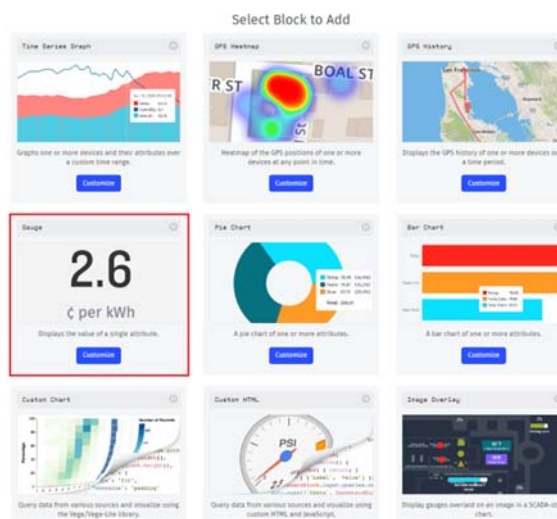
A dashboard is made up of blocks. Each block offers a different way to present data. Create a new dashboard by selecting "Dashboards" from the application menu and then select "Add Dashboard".



Name the new dashboard and press "Create Dashboard" button.



Select a block. For each block, you can configure what device state data to display within the block. For example select a Gauge Block.



BLOCK OVERVIEW

The gauge block displays a single attribute value aggregated from one or more devices. It can aggregate historical data or display the most recently received data. ([View Documentation](#))

Block Header Text

Block Description Text

DATA TYPE

Choose whether you want the block to stream data in real time, or if the data should be aggregated and/or updated only with the rest of your dashboard.

Live Stream
Great for displaying data from a single device per query. Historical data is not available. Block will update automatically when new data is received.

Historical
Great for displaying data from multiple devices and/or aggregating points. Block will update at the normal dashboard refresh rate.

DURATION

Gauge blocks can aggregate historical data or display the most recently received data. When displaying historical data, the information is aggregated together using the specified aggregator.

Duration

BLOCK DATA

Select the devices and attributes to display. Devices can be specified as a selection of devices, device tags, or both. If the duration is selected as the most recent data point and more than one device is selected, the aggregator is applied to the last data point for all selected devices.

Device IDs / Tags

Attribute

Aggregation

GAUGE STYLE

Choose your gauge type, optionally set a label, and if applicable, choose a default color and set your number display rules.

Gauge Type:

Label: Color:

Min: Max:

Display as percentage between min & max
e.g. A value of 30 will display as 50% when min is 20 and max is 40.

NUMBER DISPLAY RULES

Precision Type: Digits:

DATA TRANSFORMATION

Optionally, you may provide an [expression](#) for transforming your raw data before it appears on the graph.

Expression:

The following variables are available for use within the expression:

- `value` - The raw data point's value.
- `time` - Time of the data point in milliseconds since Epoch.
- `ctx.<variableName>` - Value of a given context variable.

CONDITIONAL COLORS

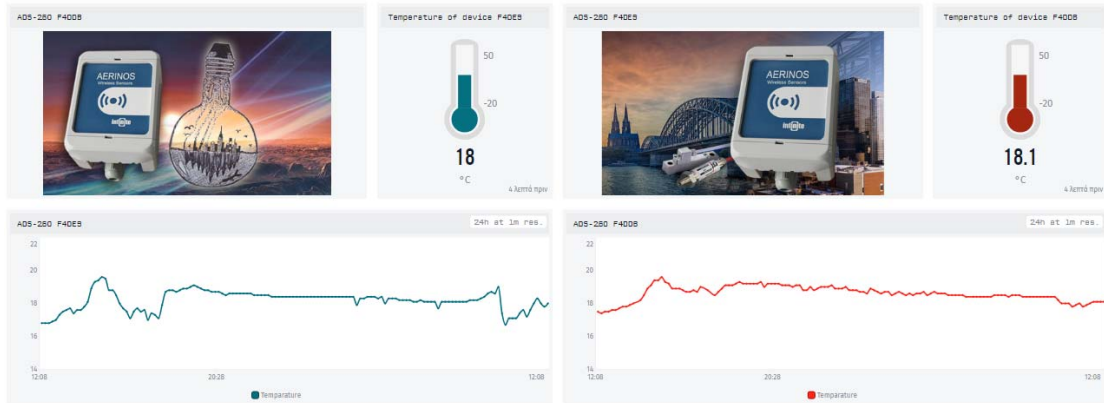
You may optionally change the gauge's color based on the query result (e.g., turning the gauge red at dangerous levels). The first [condition](#) that returns `true` determines the gauge's color. If no conditions return `true`, the default color defined above is used.

The result returned from the query is available under the conditional variable `{{value}}`, and value's percentage as it relates to the defined min and max is available under `{{percent}}`.

[Add Condition](#)



There is no limitation in the number of blocks that can be added.
The final dashboard of this project is the following.



5.6 Experiences

New application Experiences must first go through a short bootstrapping process before you starting to use the relative features.

5.6.1 Choose a Slug

YOUR APPLICATION EXPERIENCE

An experience makes it easy to build custom interfaces on top of your devices and data. Each experience includes custom users, groups, endpoints and views.

The best way to get started is with an example experience that we'll build for you. It will include an example user and a few endpoints and views. Each endpoint is powered by a workflow and rendered with a view.

To get started, give us your custom experience slug and click "Create My Experience".

Set your experience slug

Bootstrap resources

Build starter layouts and routes
This includes everything you need to authenticate users and start rendering pages.

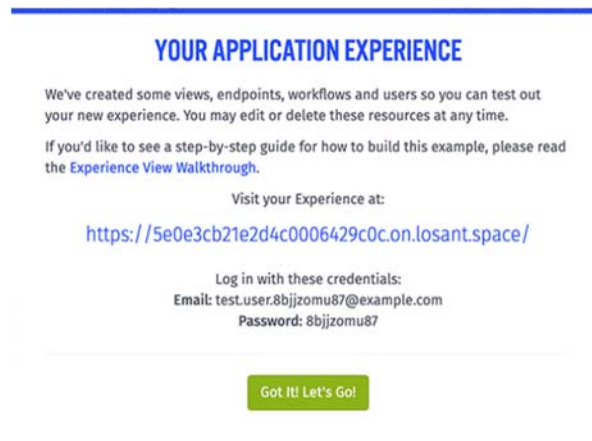
Skip bootstrapping and build from scratch
It's up to you to define your basic layouts, authentication routes and workflows.

By default, the application experience includes a slug that matches your application ID; this slug cannot be deleted. You can also enter a custom slug during the bootstrapping process.

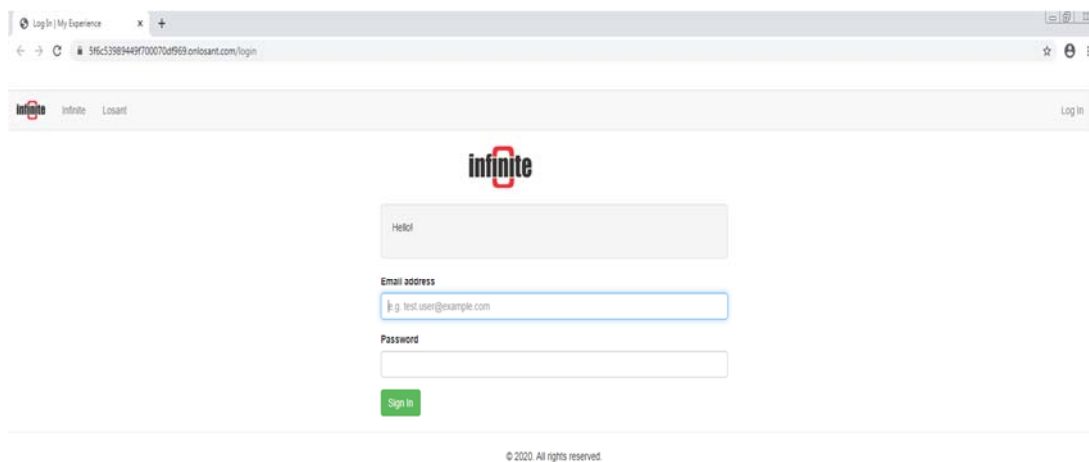
Click on "Create My Experience".

5.6.2 Test your Experience

If you chose to create the sample resources, you'll receive instructions for testing your new endpoints and views.

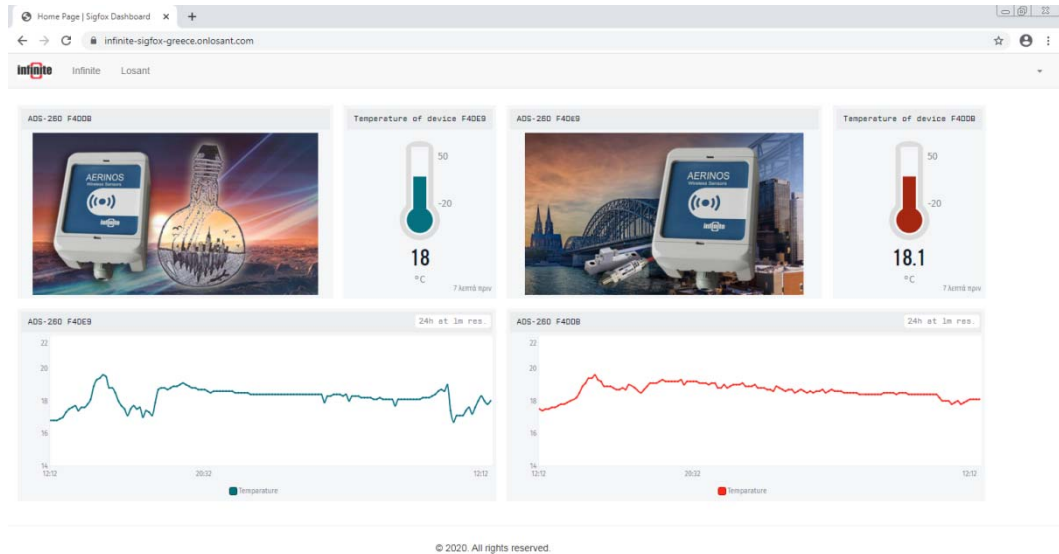


Click the link provided on the summary screen, which should redirect you to your new login page.



Sign in with the provided credentials and you will then see the placeholder home page. The page will look like that.

ADS-260 - How to build a Sigfox IoT telemetry application



Try to log in to the page we created as an example for you to get a first impression of the result.

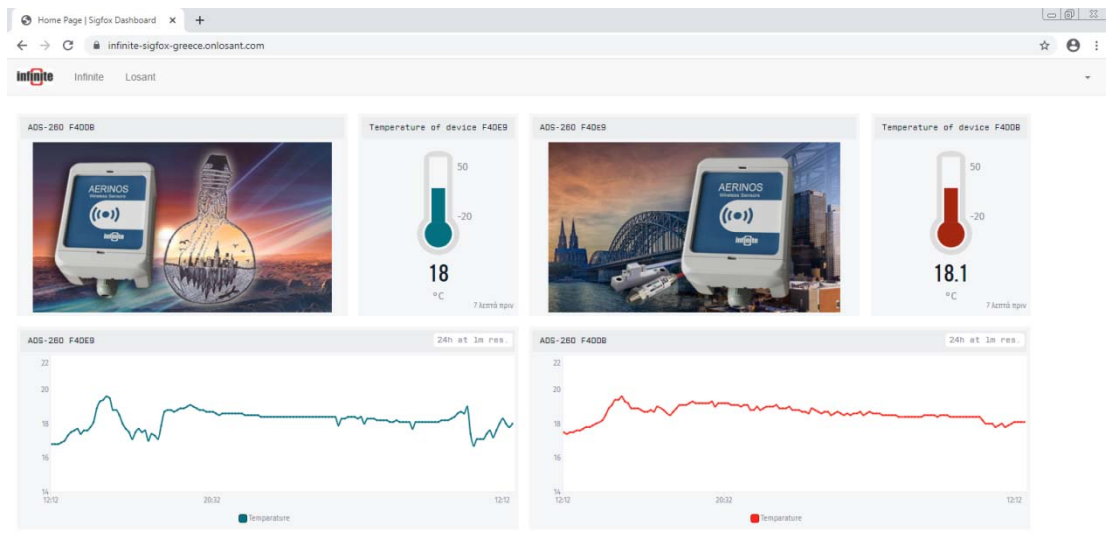
Navigate to,

<https://infinite-sigfox-greece.onlosant.com>

To log in, type "guest@infinite.com" for the email address and "Infinite" for the password.

The screenshot shows the login page of the Sigfox dashboard. The browser address bar displays infinite-sigfox-greece.onlosant.com/login. The page features the 'infinite' logo at the top center. Below the logo is a form with the following fields: a text input field containing 'Infinite 2020', an 'Email address' field containing 'guest@infinite.com', and a 'Password' field with masked characters. A green 'Sign In' button is located below the password field. The 'infinite' logo and the name 'Losant' are visible in the top navigation bar, along with a 'Log In' link. A copyright notice '© 2020. All rights reserved.' is at the bottom.

When you successfully log in, you will see the below home page.



Disclaimer:

Sigfox backend is a registered trademark of Sigfox S.A., France. All products and software of Sigfox are mentioned in this document for educational and demonstration purposes.

Losant is a registered trademark of Losant IOT, USA. All products and software of Losant are mentioned in this document for educational and demonstration purposes.

© 2020, Infinite Informatics Ltd

Infinite Informatics, Ltd

1, Valaoritou Street
GR-54626 Thessaloniki, Greece
Phone: +30-2310-553545
E: info@indinf.gr
W: www.infinite.com.gr