API V2 Reference

Contents

1 Overview

The API documentation is available at https://api.waziup.io/docs. With this website, you can explore and interact with all the endpoints of the Waziup API.

Once the preliminaries completed, head to the following sections:

- Access control
- Device management
- Notifications
- Gateways
- MQTT access

This documentation is available in PDF format[1].

2 Preliminaries

The first step is to install the "cURL" command. Curl is a very neat command to interact with REST APIs, using the HTTP protocol. If not already installed on your system, it can be found here: https://curl.haxx.se/ For Windows, it can be downloaded here[2]. Unzip the file somewhere. You can then open a "Command" window, and go in the "bin" folder than you extracted. The commands below can now be copy-pasted and executed in the command window. You can also install the jq command[3]. It allows to pretty-print JSON informations.

If your application is written in Javascript, you can use the Waziup library[4] instead of using directly the API. It's easier! All documentation is provided in the README files in the repository.

If you have an application using previous versions of the API, use the migration guide[5].

[1] /docs/WaziCloud_API_Reference-V2.1.pdf
[2] https://curl.haxx.se/windows/
[3] https://stedolan.github.io/jq/download/
[4] https://github.com/Waziup/waziup-js
[5] ./migration_guide_v2

3 Access control

This tutorial will guide you through the access control features of the Waziup API version 2, step by step. First of all, open the API documentation available at https://api.waziup.io/docs. With this website, you can explore and interact with all the endpoints of the Waziup API.

3.1 User creation

First of all, you need to create a user on the dashboard: https://dashboard.waziup.io. This will give you access to the dashboard features, and to all API features. During user creation, you need to provide:

- username: your username,
- firstName: your firstname, such as "Barack",
- lastName: your last name, such as "Obama",
- email: your email,
- phone: your phone number including country code, such as "+233248107811",
- address: your physical address,
- facebook: your facebook ID,
- twitter: your twitter account ID, without the "@".

3.2 Authentication

Once created, you can request an authentication token:

```
curl -X POST "https://api.waziup.io/api/v2/auth/token" -H "Content-Type:
    application/json" -d '{"username":"cdupont","password":"password"}'
```

This will return a token (a big number). This token can be included in all subsequent API call. This will allow you to create private resources such as devices or gateways and manage their access rights. For example, here is how to create a device belonging to the user cdupont:

```
curl -X POST "https://api.waziup.io/api/v2/devices" -H "accept:application
    /json" -H "Authorization:Bearer <token>" -H "Content-Type:application/
    json" -d '{"id": "MyDevice"}'
```

Replace the tag in the above command by the token you receive with the previous command. Once you created the device, it should be visible on the dashboard: https://dashboard.waziup.io/devices/MyDevice. Token should be used also in other calls, such as datapoint pushing. Tokens should be obtained before each request, as they are short-lived (10 minute). If a token is not valid, an error 401 "Unauthorized" will be returned. It is still possible to push devices/data without a token, however they will be created as belonging to user "guest" and visibility will be public.

It is also practical to store the token in a variable and use that variable in any other calls:

```
TOKEN=`curl -X POST "https://api.waziup.io/api/v2/auth/token" -H "Content-
    Type:application/json" -d '{"username":"cdupont","password":"password"
    }'`
curl -X POST "https://api.waziup.io/api/v2/devices" -H "accept:application
    /json" -H "Authorization:Bearer $TOKEN" -H "Content-Type:application/
    json" -d '{"id": "MyDevice"}'
```

3.3 Permissions

Permissions can be retrieve using the following command:

```
curl -X GET "https://api.waziup.io/api/v2/auth/permissions/devices" -H "
    Authorization:Bearer $TOKEN" -H "accept: application/json"
```

This will return the list of all permissions for the owner of the token:

```
[
  {
    "resource": "PublicDevice",
    "scopes": [
      "devices:view"
    ]
  },
  {
    "resource": "MyDevice",
    "scopes": [
      "devices:delete",
      "devices-data:view",
      "devices:view",
      "devices:update",
```

```
      "devices-data:create"
    ]
  }
]
```

In the example above, the user can only *view* the device "PublicDevice". He can do more on the device "MyDevice". For a device, the following access rights are possible:

- `devices:view`: the user can view the device,
- `devices:update`: the user can update the device,
- `devices:delete`: the user can delete the device,
- `devices-data:view`: the user can view the datapoints from the device,
- `devices-data:create`: the user can push additional datapoints on the device.

The access rights to a particular device are decided based on the owner, the visibility and the sharing of that device.

Ownership

Has showed above, a device can be created using a token from a specific user. This device will belong to him. The owner of a device has full privilege on it. For instance, only him (and admins) can modify or delete the device. If you don't have permission to modify or delete a device, you will receive an error "403 Forbidden".

Visibility

The API includes the concept of device visibility. Visibility can be either private or public. For instance, a gateway can create private devices this way:

```
curl -X POST "https://api.waziup.io/api/v2/devices" -H "accept:application
   /json" -H "Authorization:Bearer $TOKEN" -H "Content-Type:application/
   json" -d '{"id":"MyDevice", "visibility":"private"}'
```

In this example, we create a private device called MyDevice, under the account of cdupont (the owner of the token). Private devices can only be seen by its owner. This device will appear on the dashboard when login in with the account of the user "cdupont". Public devices can potentially be seen by everybody in the dashboard. However, public devices can only be updated or deleted by its owner or an admin.

Sharing devices

Using the dashboard, a user can decide to share a device with selected users. Sharing devices can be made in the Profile of the user. In particular, users can give the follow-

ing privileges to other users: "device:view", "device:update", "device:delete", "device-data:view" and "device-data:create".

Private devices can be shared with other users. For example, you can create a private device, and share the privilege "device:view" with another user. This will allow that user to see your private device; however he will still not be able to update or delete it. Public devices can also be shared, in the case you would like to give full privilege to another user (i.e. update or delete the device).

4 Device management

This tutorial will guide you through the Waziup API version 2, step by step. We will create a device, update its attributes and then delete it. We will perform all the commands without authentication, so the sensor created will be public. If you want to know more about private sensors, see this tutorial[6].

First of all, open the API documentation available at https://api.waziup.io/docs. With this website, you can explore and interact with all the endpoints of the Waziup API.

4.1 Create a device

We will create a device named "MyDevice".

```
curl -X POST "https://api.waziup.io/api/v2/devices" -H "Content-Type:
   application/json" -d '{"id": "MyDevice"}'
```

You just created a device! You can already go on Waziup dashboard[7] and find it among the devices.

There are 3 parts to this command:

- the method: `POST`
- the URL: `https://api.waziup.io/api/v2/devices`
- a header: `accept: application/json`
- and the data: `{"id": "MyDevice"}`

[6] ../access_control
[7] http://dashboard.waziup.io/devices/MyDevice

The HTTP protocol supports several methods. The most used are GET, PUT, POST and DELETE. They are used to act on a *resource*. The resource is identified by an URL, here `https://api.waziup.io/api/v2/devices`. The data is the content of the request. In our example, the data section contains the id of the device. Finally, the Header provide additional technical information about the request. In this case, we specify that the request data is made with JSON.

Here a more complete example:

```
curl -X POST "https://api.waziup.io/api/v2/devices" -H "Content-Type:
    application/json" -d @- <<EOF
{
  "id": "MyDevice",
  "gateway_id": "XXX",
  "name": "My weather station",
  "visibility": "public",
  "location": {
    "latitude": 5.36,
    "longitude": 4.0083
  },
  "sensors": [
    {
      "id": "TC",
      "name": "My garden temperature",
      "sensing_device": "SoilThermometer",
      "quantity_kind": "SoilTemperature",
      "unit": "DegreeCelsius",
    }
  ]
}
EOF
```

Much more complete! If you check again on the dashboard, you should see more details. We provided:

- The `id` of the device. The `id` it should be unique among all devices. It is used by physical gateway and devices to push the data (see section "push your data").
- A `gateway_id`. It is the id of the gateway to which it is connected. This allows to recognise which devices are connected to the same gateway.
- A `name` for the device. This is a more informative name that just the device `id`: For example you can give the value `my garden humidity device`.

- The `domain` of the sensor. The domain is used to group sensors from the same application domain, for example sensors from a specific farm.
- A `visibility`. Visibility can take the values "public" or "private". See the section "Visibility" for more information.

- The `location` in latitude/longitude. This is the GPS position of your device.
- The `sensors`. The sensors are the physical sensors connected to your device.

Sensors correspond to the individual physical sensors attached to your device. Each sensor have:

- an `id`. This is the id of the sensor, as used by the gateway/device to push the data. For example, "TC" refers to a temperature sensor, measuring temperature in degrees.
- a `name`. Similar to device name, this is a user-readable name for the sensor, such as "temperature".
- a `sensing_device`. This is the kind of physical sensor used: a temperature sensor, a soil moisture sensor...
- a `quantity_kind`. This is what you are really measuring with the sensor. For example, air temperature or water temperature?
- a `unit` of measurement. For example, "DegreeCelcius".

Once you performed the command, your device should be immediatly visibile on the dashboard: https://dashboard.waziup.io/devices/MyDevice.

4.2 Read a list of devices

Once created, you can read your devices. The following command allows you to read a list of all devices:

```
curl -X GET "https://api.waziup.io/api/v2/devices" -H "Accept: application
   /json"
```

Of course, a lot of different devices might be returned by this request. By default, the 20 first devices are returned. You can filter this list with the following query parameters:

- *q*: filter the list according to some criteria
- *limit*: number of entries per page
- *offset*: offset for the starting entry

The *q* parameter allows you to filter the list of devices returned using a criteria. The format for this parameter is: `q=<field>==<criteria>`. For instance, here is how you get all the devices belonging to a particular owner:

```
curl -X GET "https://api.waziup.io/api/v2/devices?q=owner==cdupont" -H "
    Accept: application/json"
```

This will return the list of devices belonging to user `cdupont`. All device fields can be used to filter the list, such as `owner`. To get more than 20 devices, you need to use the `limit` query parameter. It can be used this way:

```
curl -X GET "https://api.waziup.io/api/v2/devices?limit=100" -H "Accept:
    application/json"
```

This query will extand the number of devices returned to 100. To get the next 100 devices, add the `offset` parameter:

```
curl -X GET "https://api.waziup.io/api/v2/devices?limit=100&offset=100" -H
     "Accept: application/json"
```

This command will return the next 100 devices in the list (i.e. devices 101 to 200). To get again the next 100 devices, increase again the offset parameter to 200. Please note that the maximum number of devices that can be returned in a single command is 1000 (by setting `limit=1000`).

4.3 Read a particular device

If you want to read a single device, use this command:

```
curl -X GET "https://api.waziup.io/api/v2/devices/MyDevice" -H "Accept:
    application/json"
```

Set the URL parameter `MyDevice` to any device ID you want. This will return the full information on that particular device:

```
{
  "id": "MyDevice",
  "gateway_id": "XXX",
  "name": "My weather station",
  "location": {
```

```
    "latitude": 5.36,
    "longitude": 4.0083
  },
  "date_created": "2018-10-19T09:45:47.00Z",
  "date_modified": "2018-10-19T09:45:47.00Z",
  "owner": "cdupont",
  "visibility": "public",
  "sensors": [
    {
      "id": "TC",
      "name": "My garden temperature",
      "sensing_device": "SoilThermometer",
      "quantity_kind": "SoilTemperature",
      "unit": "DegreeCelsius",
      "last_value": {
        "value": "25.6",
        "timestamp": "2016-06-08T18:20:27.873Z",
        "date_received": "2018-10-19T10:16:20.00Z"
      }
    }
  ]
}
```

With respect to device creation, three additional fields are returned: `date_created`, `date_modified` and `last_value`. `date_created` and `date_modified` are the date at which you created your device, and the date at which you last modified the device, respectively. `last_value` contains the information about the last datapoint that you pushed to your sensor (see Section "push datapoints" below).

4.4 Update your device

Say you want to update the name of the device that you just created.

```
curl -X PUT "https://api.waziup.io/api/v2/devices/MyDevice/name" -H "
   Content-Type: text/plain" -d "My garden device"
```

Notice that we used the "PUT" method and added "/name" at the end of the URL. This device name will be updated to the value "My garden device". You can modify

individually any field in the device this way: name, gateway_id, domain, location and visibility. Similarly, sensors can be modified:

```
curl -X PUT "https://api.waziup.io/api/v2/devices/MyDevice/sensors/TC/name
    " -H "Content-Type: text/plain" -d "My garden device"
```

4.5 Create sensors

Sensors can be added individually, even after the device has been created. This is an example of a correct sensor creation:

```
curl -X POST "https://api.waziup.io/api/v2/devices/MyDevice/sensors" -H "
    accept: application/json" -H "Content-Type: application/json" -d '{ "id
    ": "SM"}'
```

This will add a single sensor called "SM" (for "Soil Moisture") to the device named "MyDevice".

4.6 Check your sensor

This is an example of a reading a single sensor:

```
curl -X GET "https://api.waziup.io/api/v2/devices/MyDevice/sensors/TC" -H
    "accept: application/json"
```

This will return the sensor called "SM" for the device named "MyDevice".


```
{
  "id": "TC",
  "name": "My garden temperature",
  "sensing_device": "SoilThermometer",
  "quantity_kind": "SoilTemperature",
  "unit": "DegreeCelsius",
  "last_value": {
    "value": "25.6",
    "timestamp": "2016-06-08T18:20:27.873Z",
    "date_received": "2018-10-19T10:16:20.00Z"
  }
}
```

If the sensor doesn't exist, an error "404" will be returned.

4.7 Push data to your sensor

You can push a new datapoint to your sensor. For example, here is how you can push the value 22.6 to sensor TC of device MyDevice:

```
curl -X POST "https://api.waziup.io/api/v2/devices/MyDevice/sensors/TC/
   values" -H "Content-Type: application/json" -d '{"value": "25.6", "
   timestamp": "2016-06-08T18:20:27.873Z"}'
```

This will add a new datapoint to your sensor. The field `timestamp` contains the exact date at which this measurement as been taken by your sensor. This field is optional.

Once you pushed the value, You should already check that your datapoint is arrived on the dashboard: https://dashboard.waziup.io/devices/MyDevice/sensors/TC.

4.8 Read datapoints

Once you pushed several datapoints to your sensor, it's time to read them. This is performed by the following command:

```
curl -X GET "https://api.waziup.io/api/v2/devices/MyDevice/sensors/TC/
   values" -H "Accept: application/json"
```

This will return the list of datapoints:

```
[
  {
    "timestamp": "2016-06-08T18:20:27.873Z",
    "value": "25.6",
    "date_received": "2018-10-19T10:16:20.000Z"
  },
  {
    "timestamp": "2016-06-20T18:20:27.873Z",
    "value": "30",
    "date_received": "2018-10-20T10:16:21.000Z"
  }
]
```

An additional field is returned: `date_received`. It contains the date at which the value was received on the Cloud platform. It can be different to `timestamp`: `timestamp` is the date at which the value was measured, while `date_received is the date at which is was received. As such`,date_received' can have a latter date if there was e.g. network connectivity problems that delayed the sending of the data.

However, there can be a huge number of datapoints registered on your sensor! You can filter those datapoints with several query parameters:

- *lastN*: This parameter allows to return X last datapoints (by timestamp date),
- *limit*: allows to limit the number of datapoints returned (similar to sensor list),
- *offset*: used in combination to `limit`, it allows to paginate the datapoints returned,
- *dateFrom*: sets a minimum timestamp date to retrieve the datapoint list,
- *dateTo*: sets a maximum timestamp date to retrieve the datapoint list.

As an example, here is how to retrieve the first 100 datapoints between two particular dates:

```
curl -X GET "https://api.waziup.io/api/v2/device/MyDevice/sensors/TC/
   values?limit=100&offset=0&dateFrom=2016-01-01T00:00:00.000Z&dateTo
   =2019-01-31T23:59:59.999Z"
```

4.9 Delete your device

Finally, let's destroy this device. Perform this command:

```
curl -X DELETE "https//api.waziup.io/api/v2/devices/MyDevice"
```

Poof! Your sensor is gone.

5 Users

The API allows you to see the list of users. This can be done using the following command:

```
curl -X GET "https://api.waziup.io/api/v2/users" -H "accept: application/
   json"
```

This command will yeld the list of users:

```
[
  {
    "id": "39575669-0fd7-4c01-8461-97252c15e540",
    "createdTime": 1530019538772,
    "username": "1234you",
    "firstName": "Richard",
    "lastName": "Ann Boakye",
    "email": "xxx@hotmail.com",
    "phone": ["+233248107811"],
    "address": ["12 Einstein road, Dakar"],
    "facebook": ["extraOrdinaire"],
    "twitter": ["xtra_ordinaire"]
  }
]
```

You can also retrieve a single user using his user ID:

```
curl -X GET "https://api.waziup.io/api/v2/users/39575669-0fd7-4c01
   -8461-97252c15e540" -H "accept: application/json"
```

6 Gateways

This tutorial will show you how to connect your gateway to Waziup. First of all, open the API documentation available at https://api.waziup.io/docs. With this website, you can explore and interact with all the endpoints of the Waziup API.

6.1 Declare the gateway

At start-up, a gateway should declare itself to the Cloud. This is done with the following command:

```
curl -X POST "https://api.waziup.io/api/v2/gateways" -H "Content-Type:
   application/json;charset=utf-8" -H "Authorization:Bearer $TOKEN" -d '{
   "visibility": "public", "name": "My gateway", "id": "GW1"}'
```

You need to choose a unique ID for the gateway. The gateway then needs to push regularly it's "heartbeat" on the /gateways/{gw_id}/health endpoint.

6.2 Simple gateway protocol

As a simple approach, a gateway can just push datapoints on existing devices on the platform. The devices and sensors should be created beforehand on the dashboard[8]. To create the device, follow this tutorial[9]. We will not use any authentication, so the sensor need to be public.

We are now ready to push a new datapoint! Follow those instructions[10] to push the datapoint. Once pushed, you should alerady see the new value displayed on the dashboard!

6.3 Complex gateway protocol

Gateways can create the devices themselves (instead of having to create them manually on the dashboard). In order to maintain privacy, a token should be obtained before each call. Once this is done, datapoints can be pushed. A suggested protocol is the following:

1. Get a token: `GET api/v2/auth/token`. See here[11].
2. Verify if the sensor exists: `GET /api/v2/devices/<device_id>`. See here[12].
3. If a 404 is received, create it: `POST /api/v2/devices`. See here[13].
4. Verify if the measurement exists: `GET /api/v2/devices/<device_id>/sensors/<sensor_id`. See here[14].
5. If a 404 is received, create it: `POST /api/v2/devices/<device_id>/sensors`. See here[15].
6. Finally, push the datapoint: `POST /api/v2/devices/<device_id>/sensors/<sensor_id>/value`. See here[16].

[8] https://dashboard.waziup.io
[9] /tutorials/software/dashboard/
[10] ../device_management/#push-data-to-your-sensor-node
[11] ../access_control/#authentication
[12] ../sensor_management/#read-a-particular-sensor
[13] ../sensor_management/#create-a-sensor-node
[14] ../sensor_management/#check-your-measurement
[15] ../sensor_management/#create-measurements
[16] ../sensor_management/#push-data-to-your-sensor-node

This protocol takes care of creating a device and sensor for the owner of the device. It then pushes the datapoint.

7 Notifications

This tutorial will guide you through the Waziup API version 2, step by step. We will create and manage notifications.

Let's imagine that we want to monitor the temperature in your fridge. If the temperature is too high, you should receive a message on your phone! First of all, let's create a device in this tutorial[17]. The device should have name "MyDevice" with a sensor called "TC1". We will monitor this temperature measurement and create a notification. This notification will send us message via SMS if the temperature goes above 10 Degree Celsius.

You should also create a user on the dashboard[18], and input your phone number with the international prefix.

7.1 Create a notification

We will create a new notification for our fridge. The following command will create the notification. An SMS message will be sent to the user `cdupont` if the temperature inside the fridge is above 10 Degree Celsius.

```
curl -X POST "https://api.waziup.io/api/v2/notifications" -H "content-type
    : application/json;charset=utf-8" -d @- <<EOF
{
  "description": "Senddd text message",
  "condition": {
    "devices": ["MyDevice"],
    "sensors": ["TC1"],
    "expression": "TC1>10"
  },
  "action": {
    "usernames": ["cdupont"],
    "channels": ["sms"],
    "message": "Warning, temperature is too high. {id} value is {TC} C"
```

[17] ../device_management
[18] https://dashboard.waziup.io

```
  },
  "expires": "2025-10-13T14:51:22.12Z",
  "throttling": 10,
}
EOF
```

The various fields are:

- `description` is the description of the notification,
- `condition` describes the condition under which the notification will be triggered,
- `action` describes which message to send, to which users and on which channels,
- `expires` is the date where this notification will not be sent anymore,
- `throttling` is the minimum delay between two notification sendings, in seconds. For example, a value of 3600 guaranties that a maximum of one message per hour will be sent to the users,
- `status` allows to activate or deactivate the notification. Possible values are `active` or `inactive`.

`condition` has the following sub-fields:

- `devices` is the list of devices under scrutiny by this notification,
- `sensors` are the sensors that will be used,
- `expression` gives the condition under which the notification will be triggered. For example, the expression `TC<40` will trigger the notification if the sensor `TC` has a value below 40 Degreee Celsius.

`action` has the following sub-fields:

- `usernames` is the list of users that will receive a message if the notification is triggered. In our example, the user `cdupont` will receive the notification messages.
- `channels` is the list of channels to where sending the notification messages. Possible values are `twitter`, `sms` and `voice`.
- `message` is the text that will be sent to the users on the selected channels. This message can contain variables: for instance `{id}` will be replaced by the sensor id, while `{<measurement_id>}` will be replaced by the value measured.

Once you executed the command above, you will get a number as a reply: this is the notification id. Keep it aside for the rest of the tutorial.

So, did you receive the SMS? You should receive the message `Warning, temperature is too high. MySensor value is 11 C`. The variables are replaced in the message sent:

{id} becomes the sensor id ("MySensor") and {TC} becomes the last value measured by the sensor.

7.2 See your notification

You can see the notification that you have just created:

```
curl -X GET "https://api.waziup.io/api/v2/notifications/<notif_id>" -H "
    accept: application/json"
```

Replace with the id returned by the previous command. You should see your notification:


```
{
  "id": "5bcb9aca2b4545c9fbdfc851",
  "description": "Send text message",
  "condition": {
    "devices": [
      "MyDevice"
    ],
    "sensors": [
      "TC"
    ],
    "expression": "TC<40"
  },
  "action": {
    "usernames": [
      "cdupont"
    ],
    "channels": [
      "sms"
    ],
    "message": "Warning, temperature is too high. {id} value is {TC} C"
  },
  "expires": "2030-10-13T14:51:22.00Z",
  "throttling": 3600,
  "status": "active",
  "times_sent": 1,
  "last_notification": "2018-10-20T21:56:58.00Z"
}
```

The notification return is identical to want you created, with two additional fields:

- `times_sent`: this shows that the notification message has been sent out once.
- `last_notification`: the date at which the last notification message was sent.

7.3 Trigger the notification

Notifications should be triggered when sensors are updated. Let's update our sensor:

```
curl -X POST "https://api.waziup.io/api/v2/devices/MyDevice/sensors/TC/
   value" -H "Content-Type: application/json" -d '{"value": "25.6", "
   timestamp": "2016-06-08T18:20:27.873Z"}'
```

Since we sent a value inferior to 40C, it should trigger the notification and send an SMS! For more information on how to create sensor and push datapoints, see this tutorial[19].

7.4 Pause/restart the notification

Once the notification created, you can pause it and restart it latter. This is done with the following command:

```
curl -X PUT "https://api.waziup.io/api/v2/notifications/<notif_id>/status"
    -H "Content-Type: text/plain" -d "inactive"
```

Possible values for this command are `active` and `inactive`.

7.5 Delete the notification

Fianlly, we can delete the notification. Issue the command:

```
curl -X DELETE "https://api.waziup.io/api/v2/notifications/<notif_id>"
```

All set. Good luck with notifications!

[19] ../sensor_management

8 MQTT

This tutorial will guide you through the access to the platform via MQTT. First, you need to install the MQTT client Mosquitto[20]. Waziup allows you to push sensor values through MQTT, and to subscribe on a sensor for changes.

8.1 PUBLISH

The first thing you can do is publishing a new value on an existing sensor. Make sure that a Device "MyDevice" exists on the dashboard, and that it is public: https://dashboard.waziup.io/devices/MyDevice Then push a new value:

```
mosquitto_pub -L "mqtt://api.waziup.io/devices/MyDevice/sensors/TC1/value"
    -m '{"value": "3"}'
```

The Device sensor "TC1" should turn Green and display your value "3".

8.2 SUBSCRIBE

Now, let's subscribe to a sensor:

```
mosquitto_sub -L "mqtt://api.waziup.io/devices/MyDevice/sensors/TC1/value"
```

In another console window, push again the value "3" as above. You should see the value arriving in you subscription:

```
{"value": "3"}
```

You can also use wildcards "+" and "#" to subscribe to several topics. "+" is a wildcard for a single level. For instance, this will subscribe to all sensors in device "MyDevice":

```
mosquitto_sub -L "mqtt://api.waziup.io/devices/MyDevice/sensors/+/value"
```

Instead, "#" can be used for multiple levels. For instance, this will subscribe to all sensors in device "MyDevice":

```
mosquitto_sub -L "mqtt://api.waziup.io/devices/MyDevice/#"
```

[20] https://mosquitto.org/download/

8.3 Security

You cannot push data to sensors to which you don't have access: they will simply not be recorded by the server. Unfortunately, the MQTT protocol doesn't define any error messages, so the server cannot inform you about the failure.

Likewise, you can subscribe to a sensor or multiple sensors, but the server will not send you the data if you don't have access.

If the device is private, you can publish data to it by giving your login/password:

```
mosquitto_pub -L "mqtt://api.waziup.io/devices/MyDevice/sensors/TC1/value"
    -m '{"value": "4"}' -u login -P password
```

The same can be done for subscription.

9 V2 Migration guide

This section will guide you through the update of your gateway or application to the new API V2 (see https://github.com/Waziup/Platform/blob/master/ChangeLog). This version brings a lot of novelties:

- Gateway registration
- Actuators support
- Projects support
- Calibration of sensors online
- Detailled permissions for devices/gateways/projects
- Better data endpoint

All endpoints are described in the various sections.

9.1 Renamings

If you have an application working with V1.1, in this version two renamings has been done:

- Everthing previously named "Sensor" has been renamed "Device".
- Everything previously named "Measurement" has been renamed "Sensor".

Indeed, the term “Device” reflects better the reality: an Arduino board.