

Taller Fiware - Smart Campus

Encuentro ANUIES-TIC Octubre 2018

1. Introducción

En la presentación de introducción que se hizo, hemos conversado sobre el impacto profundo que la Internet de las Cosas (IoT, *Internet of Things*) tendrá en prácticamente todas las actividades de nuestra vida.

También hemos mencionado que para sacar provecho de lo que los objetos *inteligentes* perciben, es necesario contar con una plataforma que permita capturar, procesar, mostrar y almacenar los datos capturados por esos objetos.

En este taller tendremos una primera exposición a **Fiware**, una plataforma abierta que permite desplegar aplicaciones *inteligentes* muy rápidamente. Específicamente, estaremos interactuando con su componente central, el **Orion Context Broker** (OCB), que es el encargado de recibir los datos enviados por los objetos de interés.

2. Objetivo

El objetivo central de este tutorial es dar una idea básica del flujo de datos típico al implementar aplicaciones inteligentes que utilicen información obtenida de diferentes medios como sensores, usuarios de dispositivos móviles, etcétera.

Los objetivos particulares son:

- Familiarizarse con los conceptos del Orion Context Broker en un entorno IoT.
- Implementar un prototipo de sensor ambiental con una tarjeta Arduino
- Capturar y desplegar en un tablero de mando las lecturas de un sensor de temperatura y humedad

Este taller es muy ambicioso e incluye algunos conceptos no triviales; siéntase en libertad de darle un primer recorrido rápido, enfocarse en los temas que le sean de mayor interés, y volver a él con más calma al terminar el taller.

El taller está compuesto de cuatro bloques:

1. Familiarización con el estándar de representación de datos NGSI y con los principales comandos que maneja el Orion Context Broker.
 - Interactuar con el OCB con mensajes REST en formato JSON
2. Implementar el prototipo de una estación de monitoreo (temperatura y humedad) con una tarjeta Arduino
 - Enviar los datos capturados por la tarjeta al OCB a través de Internet.
3. Visualizar los datos capturados en un tablero de control.
4. Conocer el ambiente de trabajo de Fiware Lab Cloud y desplegar una instancia de evaluación en ese ambiente

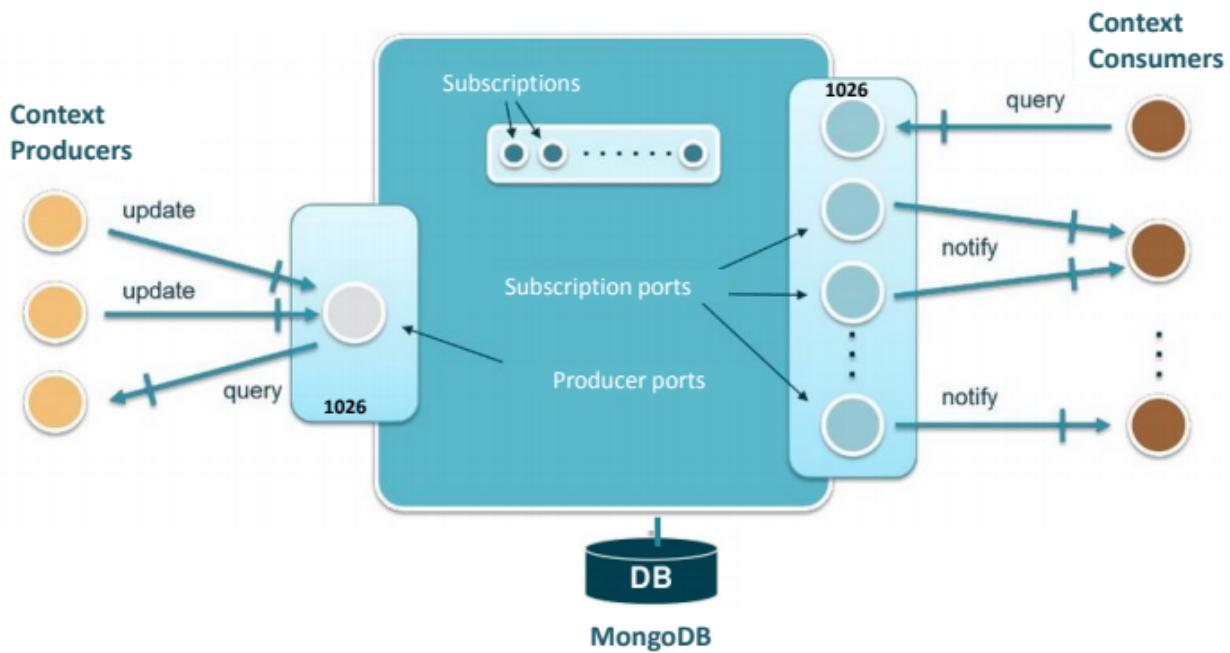
3.- Orion Context Broker

En Fiware, para que las aplicaciones puedan obtener información de los sensores y objetos inteligentes, un componente esencial es el Orion Context Broker (OCB). Orion Context Broker es una implementación de la API NGSI (*Next Generation Service Interface*) que permite manejar y asegurar la disponibilidad de la información obtenida del contexto donde se encuentra el objeto (el sensor). La versión que se utiliza actualmente es **NGSIV2**.

La especificación completa de NGSIV2 se encuentra aquí:

<http://fiware.github.io/context.Orion/api/v2/stable/>.

La interacción típica en la plataforma Fiware (como en la mayoría de las plataformas para Internet de las Cosas) consta de tres elementos: el productor de información de contexto (por ejemplo, un sensor de ocupación en un estacionamiento), un intermediario, que en nuestro caso es el OCB, y el consumidor de esa información.



El productor de información de contexto se encargará de crear nuevas entidades o de actualizar las entidades ya existentes. Típicamente accede al OCB a través del **puerto 1026**.

Los últimos datos se mantienen persistentes en el OCB con ayuda de una base de datos; actualmente se utiliza MongoDB.

El OCB funciona como intermediario entre los productores de información y otros componentes (los consumidores de información) como pueden ser un tablero de mando para representar gráficamente la información, un conector hacia bases de datos o repositorios de big data, un procesador en tiempo real, etcétera.

En este tutorial vamos a interactuar con el OCB enviando y consultando representaciones de objetos a través de una API REST.

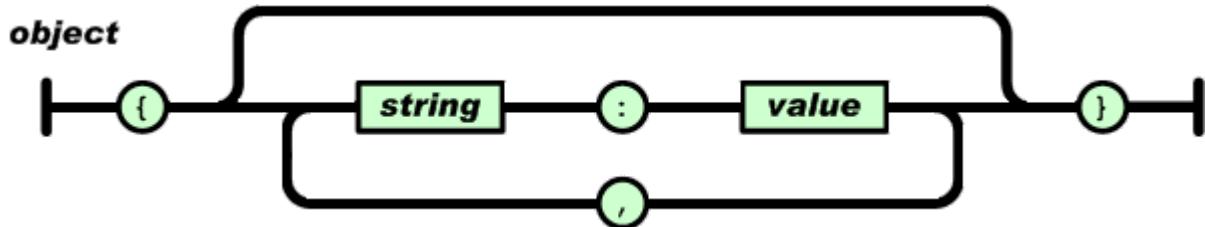
3.1 Representación de datos de contexto

Para representar objetos de la vida real se utiliza el modelo de entidades de la API NGSI. En éste se define un **modelo de datos** de información de contexto basado en *entidades* y en *atributos*. Cada entidad representa un objeto de la vida real y puede tener atributos y metadatos.

Las entidades cuentan con un identificador (ID) y un tipo. **Esta pareja ID/tipo debe ser única en el OCB.** Los atributos y metadatos se representan por una tupla **[nombre, tipo, valor]**.

Todos los datos estarán representados con el formato JSON (también pueden representarse en otro formato, por ejemplo, key/value). El formato de datos **JSON (Java Script Object Notation)** es ligero para el intercambio de información, además de ser fácil de leer, así como de procesar.

Un Objeto JSON tiene la siguiente forma:



Es decir, las propiedades del objeto se encuentran definidas entre llaves. *String* identifica una propiedad de la entidad y *value* identifica su valor. Por ejemplo, podemos representar la temperatura y la presión de una habitación con el siguiente objeto:

```
{  
    "id": "Cuarto1"  
    "type": "Room"  
    "temperature": {  
        "type": "Float",  
        "value": 23,  
        "metadata":{  
            "precision": {  
                "type": Integer,  
                "value": 3  
            }  
        }  
    },  
    "pressure":{  
        "value": 720  
    }  
}
```

PtosInCircle.png

Como se observa en el ejemplo anterior, en los atributos se puede especificar o no el tipo de dato. Se recomienda especificarlo siempre; si se omite, el OCB tratará de inferirlo.

También se observa que la metadata es opcional y en caso de existir, su formato será también una tupla [nombre, tipo, valor].

Nota: Fiware ya tiene un conjunto de modelos estandarizados. Pueden consultarse en la página <https://www.fiware.org/data-models/>. Otra página de interés es <http://schema.org/>.

Si se encuentra un modelo del objeto que deseamos representar, conviene utilizar esos esquemas para que nuestro producto sea interoperable.

Por ejemplo, si se desea administrar los **estacionamientos de un plantel**, convendría utilizar los modelos de datos de Fiware: [Parking Harmonized Data Models](#).

Dentro de esos modelos de datos, un cajón de estacionamiento puede representarse de esta manera:

```
{  
  "type": "ParkingSpot",  
  "id": "lugar01",  
  "name": "A-01",  
  "floor": "Planta baja",  
  "status": "libre",  
  "category": ["onstreet"],  
  "refParkingsite": "Universidad:EstacionamientoAlumnos"  
}
```

4. Interacciones con OCB

(Los tutoriales para empezar a trabajar con Fiware se encuentran en: <https://fiware-tutorials.readthedocs.io/en/latest/>. Este taller está inspirado en algunos de esos tutoriales.)

4.1 Introducción

El OCB contiene una interfaz tipo Web para realizar las consultas a la base de datos MongoDB. Se trata de un servicio Web de tipo REST (Representational state transfer).

En este tutorial, la interacción con el OCB se hará a través de solicitudes HTTP con un cliente REST. Para ello, se debe especificar el URL al cual estaremos haciendo la solicitud, el método REST de la solicitud, el encabezado y el cuerpo de la solicitud.

Vamos a empezar por verificar que el servidor está activo. Como se mencionó en la guía de configuración, utilizaremos el comando `GET` de HTTP que puede ser invocado con el comando `curl -X GET 'http://localhost:1026/version'`, escribiendo el URL directamente en el navegador o bien, desde nuestro cliente Insomnia REST.

```

1  {
2   "orion": {
3     "version": "1.15.0-next",
4     "uptime": "0 d, 0 h, 3 m, 19 s",
5     "git_hash": "18af270d0a496261d5e573034f744af174c5efcf",
6     "compile_time": "Sun Sep 16 12:40:32 UTC 2018",
7     "compiled_by": "root",
8     "compiled_in": "bc605f79d72d",
9     "release_date": "Sun Sep 16 12:40:32 UTC 2018",
10    "doc": "https://fiware-orion.readthedocs.org/en/master/"
11  }
12 }

```

Si todo está correcto, obtendrá una respuesta en el panel del extremo derecho de Insomnia como la siguiente:

```
{
  "orion": {
    "version": "1.15.0-next",
    "uptime": "0 d, 0 h, 3 m, 21 s",
    "git_hash": "e2ff1a8d9515ade24cf8d4b90d27af7a616c7725",
    "compile_time": "Wed Apr 4 19:08:02 UTC 2018",
    "compiled_by": "root",
    "compiled_in": "2f4a69bdc191",
    "release_date": "Wed Apr 4 19:08:02 UTC 2018",
    "doc": "https://fiware-orion.readthedocs.org/en/master/"
  }
}
```

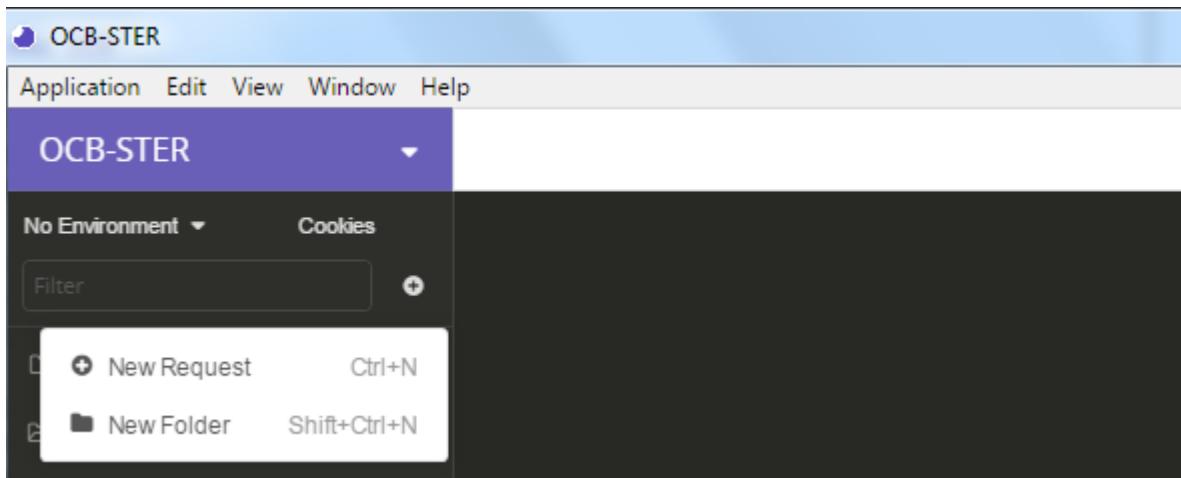
- **Nota.** Si está en Windows y no funciona el URL `http://localhost:1026`, pruebe sustituyendo `localhost` por la dirección `127.0.0.1`, la dirección IP de su computadora, o la dirección IP asignada al crear la máquina virtual con Docker (como se muestra en la figura anterior).
- Si no ha podido levantar su ambiente local, el tutor le indicará una dirección IP de un servidor externo con una imagen del OCB.

Los servicios REST permiten obtener información de forma arborecente. Es decir, es posible obtener, actualizar y borrar información de una entidad completa o valores de atributos específicos. Los métodos REST que utilizaremos son **GET, POST, PUT, DELETE, PATCH**.

El encabezado de la consulta indica en qué formato se estará recibiendo y enviando la información:

- Si la información es de tipo JSON se debe poner **application/json**
- Si es de tipo texto se debe de poner **text/plain**.
- Para indicar que se está enviando información se debe de poner **Content-Type** y para indicar que se desea recibir se debe de poner **Accept**.

Es recomendable crear una nueva carpeta en nuestro cliente REST Insomnia. Llamémosla **Operaciones-Comunes**. En esta carpeta se guardarán todas las operaciones que realizaremos en esta parte del tutorial.

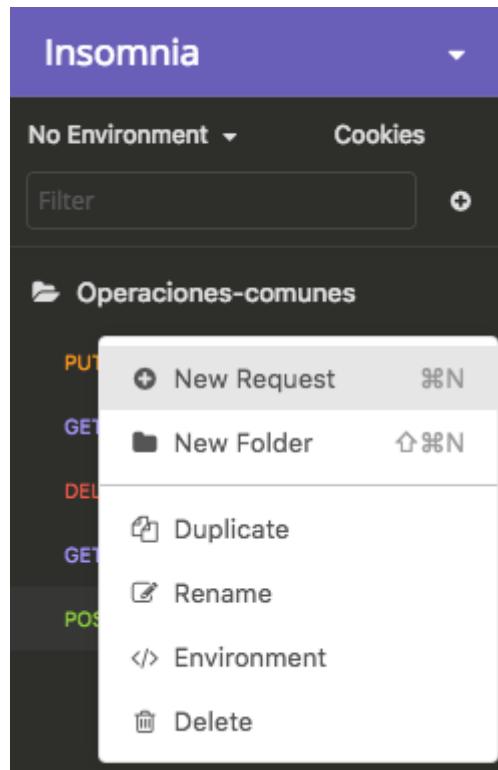


4.2 Creación de datos de contexto. Método POST

Para poder interactuar con la información de contexto en el OCB de Fiware, primero debemos crear algunos datos de contexto y enviarlos al OCB. Esto lo haremos con el método `POST` de HTTP.

Recordemos que una entidad **DEBE** tener los atributos `id` y `type`, y esta tupla debe ser única. La entidad puede tener atributos adicionales, pero cada uno debe tener, al menos un `type` y un `value`.

Comenzaremos por crear una nueva petición (New Request) en Insomnia:



El nombre sugerido para esta petición es **inserta-entidad**. La consulta tendrá el siguiente formato:

- Método: **POST**

- URL: <http://XX.XX.XX.XX:1026/v2/entities> (sustituya las xx por "localhost" o por la dirección IP que corresponda)
- BODY tipo JSON
- Header type: *application/json*. En Insomnia, el encabezado se establece automáticamente al seleccionar JSON como tipo del contenido. En algunas ocasiones utilizaremos el tipo *text/plain*.
- En el cuerpo especificaremos nuestra primera entidad en formato JSON. (Nota: Por simplicidad, estamos siendo un poco laxos con el formato de nuestras entidades. Existe un objeto "building" en los modelos de datos de Fiware).

```
{
  "id": "Classroom01",
  "type": "Room",
  "building": {
    "value": "Edif01"
  },
  "floor": {
    "value": "PB"
  },
  "status": {
    "value": "ocupado"
  },
  "temperature": {
    "type": "Float",
    "value": 23
  }
}
```

Si todo está correcto, al dar **Send** en el extremo derecho de Insomnia se debe observar el mensaje **201 CREATED** y el cuerpo de la respuesta debe estar vacío.

Insomnia		POST ▾	http://192.168.99.100:1026/v2/entities	Send	201 Created	TIME 0 ms	SIZE 0 B		
No Environment	Cookies	JSON ▾	Auth ▾	Query	Header 1	Docs	Preview ▾	Header 5	Cookie
<pre> 1 <! 2 "id": "Classroom01", 3 "type": "Room", 4 "building": { 5 "value": "Edif01" 6 }, 7 "floor": { 8 "value": "PB" 9 }, 10 "status": { 11 "value": "ocupado" 12 }, 13 "temperature": { 14 "type": "Float", 15 "value": 23 16 } 17 > </pre>					No body returned for response				
Anuies <ul style="list-style-type: none"> GET obtén todas las entidades POST Inserta Entidad GET Revisa estado del OCB (C...) PUT My Request GET GetTL1 POST MiEntidad 									

EJERCICIO. Agregue nuevas entidades con los siguientes valores:

- Como identificador, **ClassroomXY**. Sustituya "X" por el número de fila y "Y" por el número de columna de su posición en el aula. *Recuerde que este identificador debe ser único en su OCB o en el OCB compartido por todos.*
- Elija un valor entre "Edif01", "Edif02" y "Edif03" para el atributo *building*; un valor entre "PB", "1", "2" para el atributo *floor*; una temperatura entre 14.0 y 36.5 para el atributo *temperature* y un valor "ocupado", "libre", "desconocido" para el atributo *status*.

4.3 Consultas. Método GET

Para obtener información de la base de datos en el OCB se utiliza el método **GET**.

En Insomnia, es posible duplicar la consulta anterior y renombrarla. Hágalo así y nombre la nueva consulta `obten-todas-entidades`. Por supuesto, debe modificar el método de POST a GET.

Para el método GET, sólo se especifica el URL, sin Body ni Content-type. En nuestra primer consulta pediremos todas las entidades almacenadas en el OCB hasta ahora. Para ello, el URL que se utiliza es:

`http://xx.xx.xx.xx:1026/v2/entities` :

The screenshot shows the Insomnia REST Client interface. On the left, there's a sidebar with various API endpoints listed under 'Anuies'. The main area shows a GET request to 'http://192.168.99.100:1026/v2/entities'. The response status is '200 OK', time is '0 ms', and size is '258 B'. The response body is a JSON object representing a classroom entity:

```

1  [
2   {
3     "id": "Classroom01",
4     "type": "Room",
5     "building": {
6       "type": "Text",
7       "value": "Edif01",
8       "metadata": {}
9     },
10    "floor": {
11      "type": "Text",
12      "value": "PB",
13      "metadata": {}
14    },
15    "status": {
16      "type": "Text",
17      "value": "ocupado",
18      "metadata": {}
19    },
20    "temperature": {
21      "type": "Float",
22      "value": 23,
23      "metadata": {}
24  }

```

Consultas acotadas.

Podemos consultar una sola entidad agregando el identificador de esa entidad al final del URL.

EJERCICIO. Agregue una nueva consulta a Insomnia. Nómbrela `obten-una-entidad` y modifíquela para obtener únicamente una de las entidades que usted creó.

The screenshot shows the OCB-STER interface. In the top bar, it says "OCB-STER – obtén-una-entidad". The menu bar includes Application, Edit, View, Window, Tools, and Help. Below the menu is a toolbar with "OCB-STER" dropdown, "GET" button, and "Send" button. The main area has tabs for "Body", "Auth", "Query", "Header 1", and "Docs". On the left, there's a sidebar with "No Environment" dropdown, "Cookies" button, "Filter" input, and a list of "GET ConsultaDeZona". The "Body" tab shows the JSON response:

```
1+ {
2 "id": "Classroom02",
3 "type": "Room",
4+ "building": {
5   "type": "Text",
6   "value": "Edificio A"
7 }
```

A partir de la versión 2 de NGSI es posible realizar consultas (u otros métodos como PUT y DELETE) a atributos específicos de las entidades ampliando el URL:

```
GET URL/v2/entities/{entityID}/attrs/{attrName}
```

Por ejemplo, para ver el atributo "temperature" de la entidad Classroom01, se utiliza el URL

```
http://xx.xx.xx:1026/v2/entities/Room01/attrs/temperature/ ,
```

y si se desea únicamente su valor, se extiende el URL hasta:

```
http://xx.xx.xx:1026/v2/entities/Room01/attrs/temperature/value/ .
```

EJERCICIO. Agregue una nueva consulta en Insomnia.

- Nómbrala `consulta-por-atributos` y modifíquela para consultar, primero el atributo *floor* y después el atributo *status* de alguna de las entidades que usted creó.
- Los resultados obtenidos deben ser similares a éste:

```
{
  "type": "Text",
  "value": "2",
  "metadata": {}

}
-----
{
  "type": "Text",
  "value": "ocupado",
  "metadata": {}

}
```

4.4. Actualización de valores. Métodos PUT y PATCH

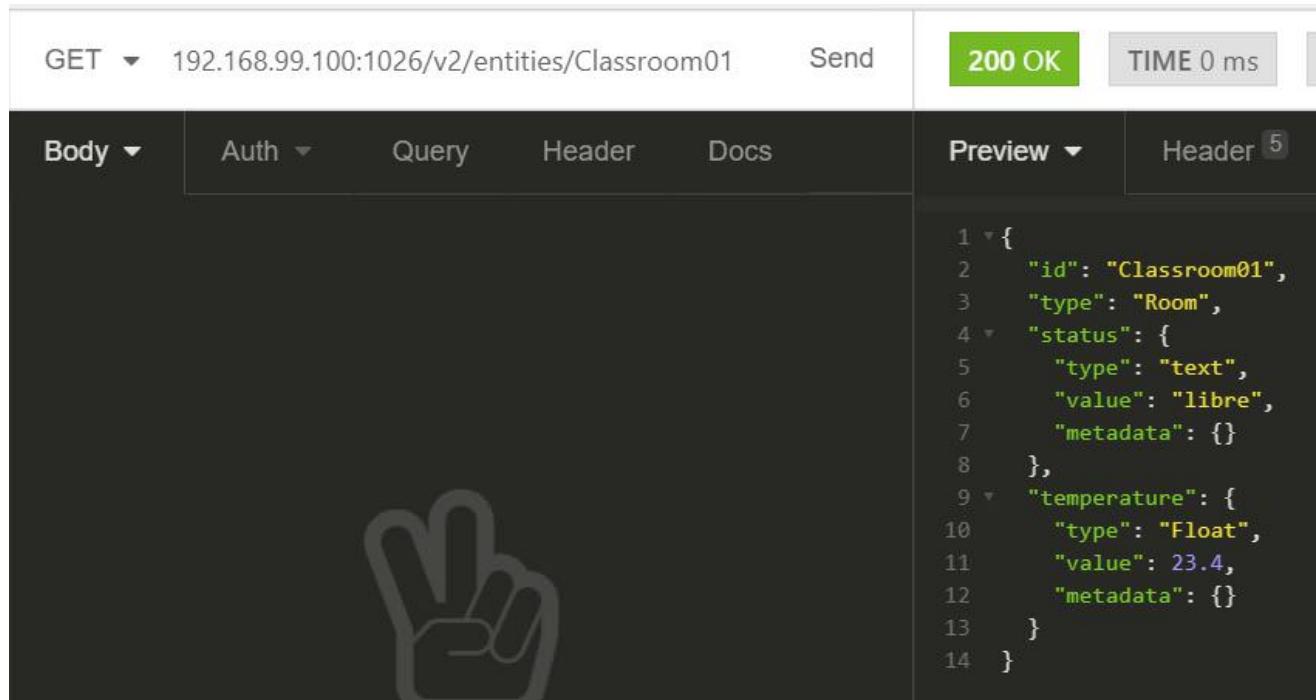
Si deseamos actualizar los valores de **TODOS** los atributos de una entidad que ya se encuentra en el OCB, se utiliza el método **PUT**.

Cuando se actualizan los valores de varios atributos a la vez, se utiliza el URL hasta el identificador de la entidad y en el cuerpo se especifican los nuevos valores en formato JSON.

En el siguiente ejemplo, se modificarán únicamente los valores de los atributos *status* y *temperature* de la entidad *Classroom01*:

```
http://xx.xx.xx.xx:1026/v2/entities/classroom01/attrs/  
Método: PUT  
Headers: Content-Type:application/json  
Body:  
{  
    "status":{  
        "value":"libre",  
        "type":"text"  
    },  
    "temperature":{  
        "type":"Float",  
        "value":23.4  
    }  
}
```

Al verificar si se realizaron los cambios, volvemos a ejecutar el método GET y veremos que **posiblemente eso NO ERA lo que esperábamos**:



GET ▾	192.168.99.100:1026/v2/entities/Classroom01	Send	200 OK	TIME 0 ms		
Body ▾	Auth ▾	Query	Header	Docs	Preview ▾	Header 5
					1 * { 2 "id": "Classroom01", 3 "type": "Room", 4 "status": { 5 "type": "text", 6 "value": "libre", 7 "metadata": {} 8 }, 9 "temperature": { 10 "type": "Float", 11 "value": 23.4, 12 "metadata": {} 13 } 14 }	

¡Hemos perdido los atributos *floor* y *building*! Efectivamente, con el método **PUT**, si se omite un atributo, éste desaparece de la entidad.

Para recuperar los atributos perdidos (y para agregar nuevos atributos), podemos utilizar el método **POST**. En el siguiente ejemplo, vamos a recuperar los atributos *floor* y *building*, cambiaremos el atributo *status* a ocupado, y agregaremos un nuevo atributo *numPersonas* con valor de 15:

```
Método: POST  
URL: http://xx.xx.xx.xx:1026/v2/entities/classroom01/attrs  
Body:
```

```
{  
    "floor": {  
        "value": "PB"  
    },  
    "building": {  
        "value": "Edif01"  
    },  
    "NumPersonas": {  
        "value": 15  
    }  
}
```

Header: Content-type: application/json

Si lo que se desea es actualizar únicamente alguno o algunos de los atributos, el método que debe usarse es **PATCH**. Por ejemplo, si sólo se desea actualizar *temperature* de Classroom01, la consulta se hará así:

Método: PATCH

URL: <http://xx.xx.xx.xx:1026/v2/entities/Classroom01/attrs>

Body:

```
{  
    "temperature": {  
        "value": 22.2,  
        "type": "Float"  
    }  
}
```

Header: Content-type: application/json

4.5. Remover entidades y atributos. Método Delete

El método DELETE permite eliminar entidades y atributos. Como ya lo imaginará, el alcance se especifica en el URL:

- Para borrar un atributo el formato es:

```
http://xx.xx.xx.xx:1026/v2/entities/{id}/attrs/{value}
```

- Para eliminar una entidad completa, se utiliza la siguiente expresión:

```
http://xx.xx.xx.xx:1026/v2/entities/{id}
```

Para probar este método, empecemos por crear una nueva entidad:

```
{
  "id": "ANUIES",
  "type": "prueba",
  "temp": {
    "value": 24,
    "type": "integer"
  },
  "NumGente": {
    "type": "integer",
    "value": 607
  }
}
```

Ejercicio

- Verifique que la entidad fue creada (utilice la consulta obten-todas-entidades).
- Elimine la variable NumGente utilizando el comando DELETE en esta URL
`http://xx.xx.xx.xx:1026/v2/entities/ANUIES/attrs/NumGente` *Este método no requiere de BODY.*
- Verifique que la variable fue eliminada.
- Ahora elimine la entidad completa con la URL `http://xx.xx.xx.xx:1026/v2/entities/ineel`.

4.6. Operaciones por lotes

NGSIv2 tiene la función `update` que permite crear, actualizar o borrar varias entidades en una sola invocación de un método POST. Por supuesto, en el cuerpo del método se especifica la representación de las entidades y sus atributos.

Hasta ahora hemos utilizado el formato JSON; en el siguiente ejemplo representaremos las entidades y atributos como **tuplas key-value**, lo cual se señalará con una opción en el URL.

La representación es más corta y fácil de leer, pero se pierde riqueza semántica. Por ejemplo, ya no se puede especificar el tipo de datos de los atributos; NGSI tratará de deducir de qué tipo se trata.

Cree una nueva operación, (nómbrela Agrega por lotes) con las siguientes características (**Observe el formato especial del URL**):

Por supuesto, si está utilizando el OCB en la nube, sólo un equipo puede hacer la operación con los identificadores del ejemplo. Si lo desea, modifique el cuerpo asignando identificadores propios

```
URL: http://xx.xx.xx.xx:1026/op/update
Método: POST
Header: Content-type: application/json
Body:
{
  "actionType": "APPEND",
  "entities": [
    {
      "id": "ClassroomB1",
      "type": "Room",
      "building": {
```

```
        "value": "Biblioteca"
    },
    "floor" :{
        "value": "3"
    },
    "temperature": {
        "type": "Float",
        "value": 26
    },
    "status":{
        "value": "libre"
    }
},
{
    "id": "ClassroomB2",
    "type": "Room",
    "building": {
        "value": "Biblioteca"
    },
    "floor" :{
        "value": "3"
    },
    "temperature": {
        "type": "Float",
        "value": 25.5
    },
    "status":{
        "value": "ocupado"
    }
},
{
    "id": "ClassroomB3",
    "type": "Room",
    "building": {
        "value": "Biblioteca"
    },
    "floor" :{
        "value": "3"
    },
    "temperature": {
        "type": "Float",
        "value": 25.0
    },
    "status":{
        "value": "libre"
    }
},
{
    "id": "ClassroomB4",
    "type": "Room",
    "building": {
        "value": "Biblioteca"
    },

```

```

"floor" :{
    "value":"3"
},
"temperature": {
    "type": "Float",
    "value": 24.5
},
"status":{
"value":"ocupado"
}
},
{
"id": "ClassroomB5",
"type": "Room",
"building": {
"value":"Biblioteca"
},
"floor" :{
    "value":"3"
},
"temperature": {
    "type": "Float",
    "value": 24.0
},
"status":{
"value":"libre"
}
},
{
"id": "ClassroomB6",
"type": "Room",
"building": {
"value":"Biblioteca"
},
"floor" :{
    "value":"3"
},
"temperature": {
    "type": "Float",
    "value": 23.5
},
"status":{
"value":"ocupado"
}
}
]
}

```

Verifique que las entidades se cargaron en el OCB.

4.7. OCB Simple Query Language

NGSI ofrece una sintaxis simplificada para filtrar información con base en algún criterio. Se pueden agregar condiciones de filtrado con el operador "&". Los resultados que se devuelven son las entidades que cumplan con TODOS los criterios.

Todas las consultas se hacen con el método GET

Al realizar una consulta, el OCB entrega por default 20 entradas. Si se desea traer más o menos, se puede agregar el parámetro `limit` a la consulta. También se puede especificar el parámetro `offset` para indicar a partir de qué entidad se obtendrán los resultados.

Por ejemplo, la siguiente consulta mostrará tres entidades de tipo "Room" (por ahora todas nuestras entidades son de ese tipo) a partir del 5º registro y éstas se mostrarán en formato key-value:

```
GET http://xx.xx.xx.xx:1026/v2/entities?limit=3&offset=5&type=Room&options=keyvalues
```

En el lienzo central, Insomnia tiene una pestaña de `query`. Podemos ver en ella cómo se va formando la consulta con los parámetros agregados al URL. Para la consulta anterior, el resultado y los campos de `query` se muestran en la siguiente figura:

The screenshot shows the Insomnia REST Client interface. On the left, there's a sidebar with a purple header and some icons. The main area has a dark background with light-colored UI elements. At the top, there's a search bar with the text "GET -3&offset=5&type=Room&options=keyValues". To the right of the search bar are buttons for "Send", "200 OK", "TIME 16 ms", and "SIZE 320". Below the search bar, there are tabs: "Body", "Auth", "Query" (which is selected), "Header", and "Docs". Under the "Query" tab, there's a "URL PREVIEW" section containing the full query URL: "http://192.168.99.100:1026/v2/entities?limit=3&offset=5&type=Room&options=keyValues". Below this, there are two input fields: "New name" and "New value", each with a gear icon. On the right side of the interface, there are tabs for "Preview", "Header" (with a count of 5), and "Cookie". The "Preview" tab displays the JSON response from the server. The response consists of three objects, each representing a room entity:

```
1 [  
2 {  
3   "id": "ClassroomB3",  
4   "type": "Room",  
5   "building": "Biblioteca",  
6   "floor": "3",  
7   "status": "libre",  
8   "temperature": 25  
9 },  
10 {  
11   "id": "ClassroomB4",  
12   "type": "Room",  
13   "building": "Biblioteca",  
14   "floor": "3",  
15   "status": "ocupado",  
16   "temperature": 24.5  
17 },  
18 {  
19   "id": "ClassroomB5",  
20   "type": "Room",  
21   "building": "Biblioteca",  
22   "floor": "3",  
23   "status": "libre",  
24   "temperature": 24  
25 }
```

Se pueden filtrar las consultas a partir del valor de algún atributo con la opción `q` (o el valor de un metadato con la opción `mq`). Por ejemplo, la siguiente consulta muestra todas las entidades en las que el atributo `temperature` es mayor o igual a 24.0 grados. Con el parámetro `attrs` se pueden especificar qué atributos se desea desplegar.

```
http://xx.xx.xx.xx:1026/v2/entities?  
q=temperature>24.0&options=keyValues&attrs=temperature,status
```

GET &options=keyValues&atrrs=temperature,status Send

200 OK

TIME 0 ms

SIZE

Body	Auth	Query	Header	Docs	Preview	Header 5	C
					<pre>1 [2 { 3 "id": "Classroom11", 4 "type": "Room", 5 "temperature": 24.5, 6 "status": "libre" 7 }, 8 { 9 "id": "ClassroomB1", 10 "type": "Room", 11 "temperature": 26, 12 "status": "libre" 13 }, 14 { 15 "id": "ClassroomB2", 16 "type": "Room", 17 "temperature": 25.5, 18 "status": "ocupado" 19 }, 20 { 21 "id": "ClassroomB3", 22 "type": "Room", 23 "temperature": 25, 24 "status": "libre"</pre>		

Select a body type from above

EJERCICIO Muestre todas las entidades en las que la temperatura es menor de 23.5 grados. Debe mostrar únicamente la temperatura.

4.8. Datos geo-referenciados

NGSI incluye el tipo de dato `geo` para expresar atributos geométricos (`geo:point`, `geo:box`, `geo:line`, etcétera). En particular, el tipo `geo:point` se utiliza frecuentemente para representar una *coordenada geográfica*, en lugar de su representación clásica de longitud y latitud.

En el tipo `geo:point` la ubicación se define con una tupla de dos números separados por coma. El primero es la latitud y el segundo la longitud. Sólo se permite la notación decimal, es decir, no se reconoce una coordenada en grados, minutos y segundos.

Para representar una ubicación, la entidad debe tener un atributo como *position*, *coordinates*, *location*, etcétera. Se puede especificar únicamente un atributo de ubicación por entidad.

Para este tutorial hemos definido una serie de puntos geo-referenciados dentro de la zona de Coyoacán en la Ciudad de México. Simulan ser puntos con generadores de energía limpia.

Con el método `POST` y la operación `update`, cargue las siguientes entidades a su OCB. (**Si se está utilizando el OCB en la nube, sólo un equipo puede hacerlo.**)

```
{
  "actionType": "APPEND",
  "entities": [
    {
      "id": "medMG1",
      "type": "microGenerator",
      "category": {
        "type": "Text",
        "value": "eolic"
      },
      "location": {
        "type": "geo:point",
        "value": "19.35216, -99.16053"
      }
    },
    {
      "id": "medMG2",
      "type": "microGenerator",
      "category": {
        "type": "Text",
        "value": "solar",
        "metadata": {}
      },
      "location": {
        "type": "geo:point",
        "value": "19.35307, -99.16304",
        "metadata": {}
      }
    },
    {
      "id": "medMG3",
      "type": "microGenerator",
      "category": {
        "type": "Text",
        "value": "solar",
        "metadata": {}
      },
      "location": {
        "type": "geo:point",
        "value": "19.35093, -99.16384",
        "metadata": {}
      }
    },
    {
      "id": "medMG4",
      "type": "microGenerator",
      "category": {
        "type": "Text",
        "value": "eolic",
        "metadata": {}
      },
      "location": {
        "type": "geo:point",
        "value": "19.35216, -99.16053"
      }
    }
  ]
}
```

```
        "value": "19.34931, -99.16693",
        "metadata": {}
    }
},
{
    "id": "medMG5",
    "type": "microGenerator",
    "category": {
        "type": "Text",
        "value": "solar",
        "metadata": {}
    },
    "location": {
        "type": "geo:point",
        "value": "19.35022, -99.15789",
        "metadata": {}
    }
},
{
    "id": "medMG6",
    "type": "microGenerator",
    "category": {
        "type": "Text",
        "value": "solar",
        "metadata": {}
    },
    "location": {
        "type": "geo:point",
        "value": "19.34823, -99.16341",
        "metadata": {}
    }
},
{
    "id": "medMG7",
    "type": "microGenerator",
    "category": {
        "type": "Text",
        "value": "eolic",
        "metadata": {}
    },
    "location": {
        "type": "geo:point",
        "value": "19.35279, -99.16579",
        "metadata": {}
    }
},
{
    "id": "medMG8",
    "type": "microGenerator",
    "category": {
        "type": "Text",
        "value": "solar",
        "metadata": {}
    }
}
```

```
        },
        "location": {
            "type": "geo:point",
            "value": "19.35178, -99.16841",
            "metadata": {}
        }
    },
    {
        "id": "medMG9",
        "type": "microGenerator",
        "category": {
            "type": "Text",
            "value": "solar",
            "metadata": {}
        },
        "location": {
            "type": "geo:point",
            "value": "19.35562, -99.15982",
            "metadata": {}
        }
    },
    {
        "id": "medMG10",
        "type": "microGenerator",
        "category": {
            "type": "Text",
            "value": "eolic",
            "metadata": {}
        },
        "location": {
            "type": "geo:point",
            "value": "19.35672, -99.1674",
            "metadata": {}
        }
    },
    {
        "id": "medMG11",
        "type": "microGenerator",
        "category": {
            "type": "Text",
            "value": "solar",
            "metadata": {}
        },
        "location": {
            "type": "geo:point",
            "value": "19.35526, -99.16796",
            "metadata": {}
        }
    },
    {
        "id": "medMG12",
        "type": "microGenerator",
        "category": {
```

```
        "type": "Text",
        "value": "eolic",
        "metadata": []
    },
    "location": {
        "type": "geo:point",
        "value": "19.34937, -99.17075",
        "metadata": []
    }
},
{
    "id": "medMG13",
    "type": "microGenerator",
    "category": {
        "type": "Text",
        "value": "eolic",
        "metadata": []
    },
    "location": {
        "type": "geo:point",
        "value": "19.35435, -99.15538",
        "metadata": []
    }
},
{
    "id": "medMG14",
    "type": "microGenerator",
    "category": {
        "type": "Text",
        "value": "solar",
        "metadata": []
    },
    "location": {
        "type": "geo:point",
        "value": "19.35726, -99.16337",
        "metadata": []
    }
},
{
    "id": "medMG15",
    "type": "microGenerator",
    "category": {
        "type": "Text",
        "value": "eolic",
        "metadata": []
    },
    "location": {
        "type": "geo:point",
        "value": "19.35086, -99.15487",
        "metadata": []
    }
},
{

```

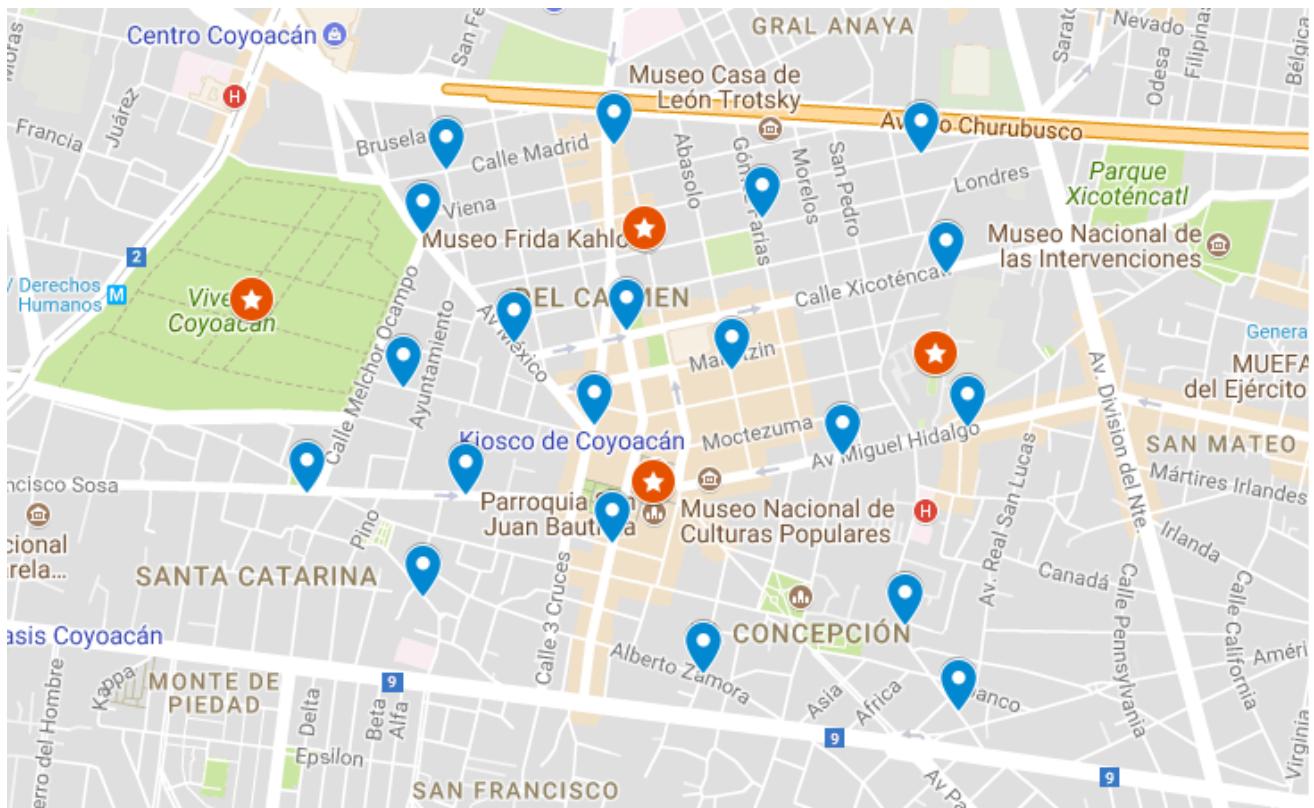
```
        "id": "medMG16",
        "type": "microGenerator",
        "category": {
            "type": "Text",
            "value": "solar",
            "metadata": {}
        },
        "location": {
            "type": "geo:point",
            "value": "19.34702, -99.16796",
            "metadata": {}
        }
    },
    {
        "id": "medMG17",
        "type": "microGenerator",
        "category": {
            "type": "Text",
            "value": "eolic",
            "metadata": {}
        },
        "location": {
            "type": "geo:point",
            "value": "19.34635, -99.15635",
            "metadata": {}
        }
    },
    {
        "id": "medMG18",
        "type": "microGenerator",
        "category": {
            "type": "Text",
            "value": "solar",
            "metadata": {}
        },
        "location": {
            "type": "geo:point",
            "value": "19.34528, -99.16122",
            "metadata": {}
        }
    },
    {
        "id": "medMG19",
        "type": "microGenerator",
        "category": {
            "type": "Text",
            "value": "solar",
            "metadata": {}
        },
        "location": {
            "type": "geo:point",
            "value": "19.35706, -99.15598",
            "metadata": {}
        }
    }
]
```

```

        }
    },
    {
        "id": "medMG20",
        "type": "microGenerator",
        "category": [
            {
                "type": "Text",
                "value": "solar",
                "metadata": {}
            },
            {
                "location": {
                    "type": "geo:point",
                    "value": "19.34443, -99.15508",
                    "metadata": {}
                }
            }
        ]
    }
]
}

```

En la siguiente figura se muestran los puntos simulados en azul, junto con algunos puntos de interés en naranja.



Ahora buscaremos lugares de interés con relación a un objeto geográfico.

Por ejemplo, en la siguiente consulta se buscarán localidades microgeneradoras que estén como máximo a 800 metros de los Viveros de Coyoacán representados por el punto `19.3538, -99.17208`.

Método: GET

URL: <http://xx.xx.xx.xx:1026/v2/entities?>

type=microGenerator&georel=near;maxDistance:800&geometry=point&coords=19.3538,-99.17208

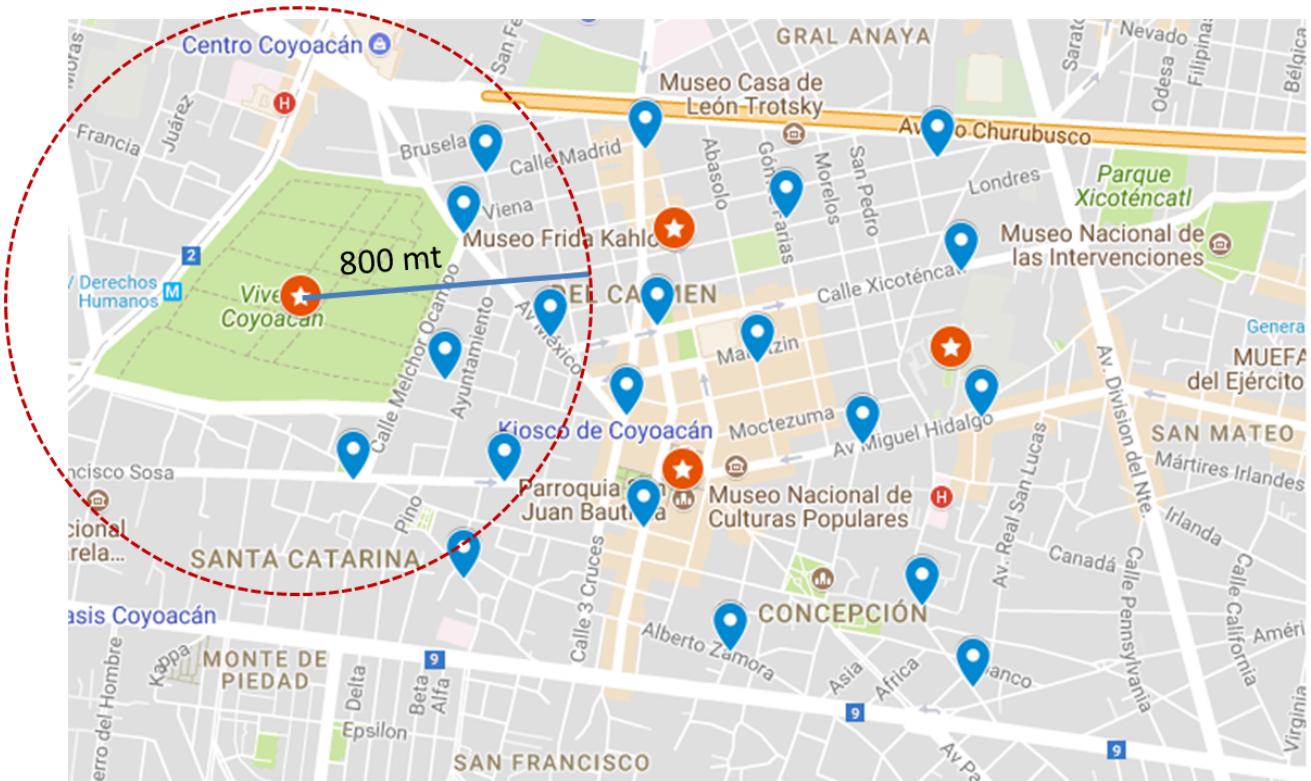
Body: Vacío

Se están buscando las entidades tipo "microGenerator" que estén como máximo 800 metros del punto especificado.

El resultado son los siguientes seis puntos:

```
[  
  {  
    "id": "medMG4",  
    "type": "microGenerator",  
    "category": {  
      "type": "Text",  
      "value": "eolic",  
      "metadata": {}  
    },  
    "location": {  
      "type": "geo:point",  
      "value": "19.34931, -99.16693",  
      "metadata": {}  
    }  
  },  
  {  
    "id": "medMG7",  
    "type": "microGenerator",  
    "category": {  
      "type": "Text",  
      "value": "eolic",  
      "metadata": {}  
    },  
    "location": {  
      "type": "geo:point",  
      "value": "19.35279, -99.16579",  
      "metadata": {}  
    }  
  },  
  {  
    "id": "medMG8",  
    "type": "microGenerator",  
    "category": {  
      "type": "Text",  
      "value": "solar",  
      "metadata": {}  
    },  
    "location": {  
      "type": "geo:point",  
      "value": "19.35178, -99.16841",  
      "metadata": {}  
    }  
  }]
```

```
        "metadata": {}
    },
},
{
    "id": "medMG10",
    "type": "microGenerator",
    "category": {
        "type": "Text",
        "value": "eolic",
        "metadata": {}
    },
    "location": {
        "type": "geo:point",
        "value": "19.35672, -99.1674",
        "metadata": {}
    }
},
{
    "id": "medMG11",
    "type": "microGenerator",
    "category": {
        "type": "Text",
        "value": "solar",
        "metadata": {}
    },
    "location": {
        "type": "geo:point",
        "value": "19.35526, -99.16796",
        "metadata": {}
    }
},
{
    "id": "medMG12",
    "type": "microGenerator",
    "category": {
        "type": "Text",
        "value": "eolic",
        "metadata": {}
    },
    "location": {
        "type": "geo:point",
        "value": "19.34937, -99.17075",
        "metadata": {}
    }
}
]
```



Ejercicio. Identifique los puntos que están dentro de un radio de 600 metros del Museo Frida Kahlo. sus coordenadas son: 19.35544, -99.16264

En el siguiente ejemplo, haremos una geo-cerca: Buscaremos los micro-generadores que se encuentran dentro del polígono formado por: Los Viveros de Coyoacán, el Museo Frida Kahlo, y el Museo Nacional de Culturas Populares. Utilizaremos la `georel` "coveredBy". Observe que en NGSI, un polígono tiene al menos cuatro coordenadas y que la primera y la última deben ser la misma.

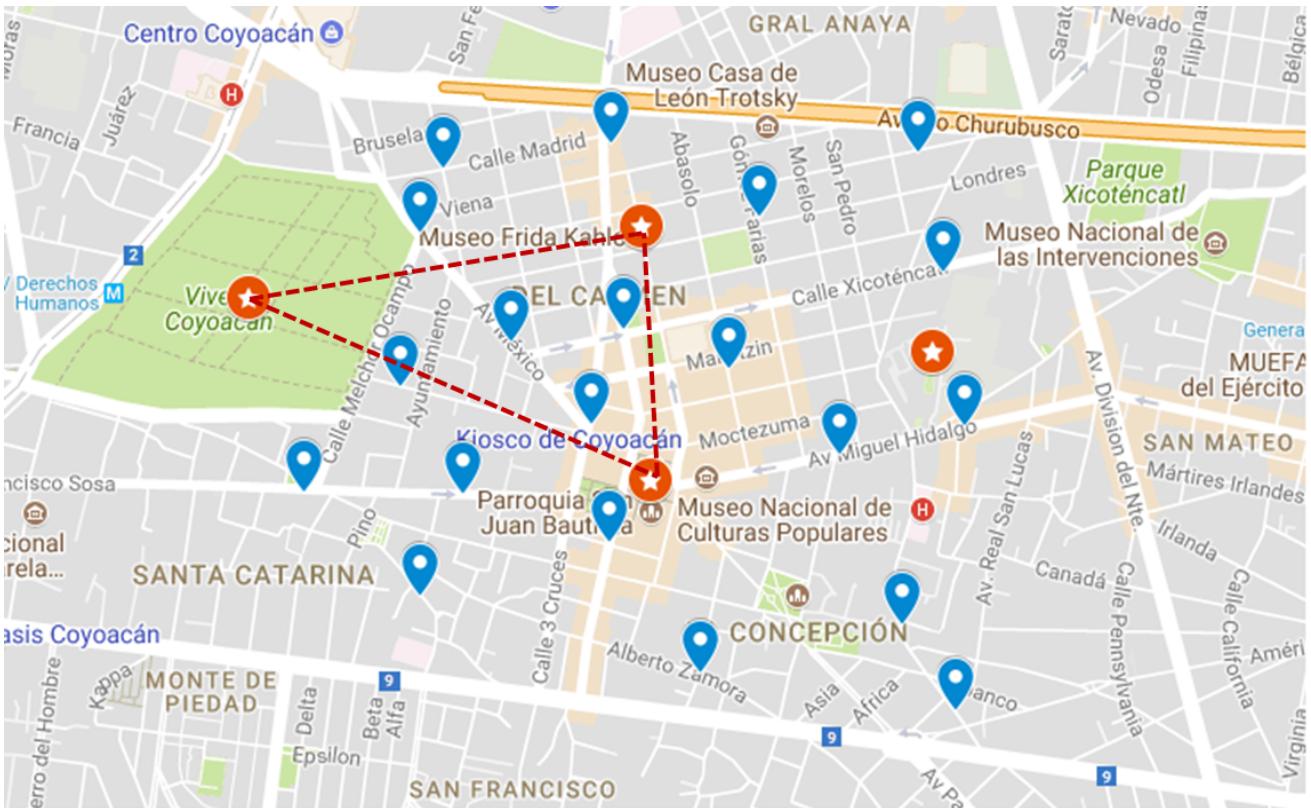
Método: GET

URL: <http://xx.xx.xx.xx:1026/v2/entities?>
`type=`microGenerator`&georel=`coveredBy`&geometry=polygon&coords=`19.3538,-99.17208;19.35544,-99.16264;19.34878,-99.16248;19.3538,-99.17208

El resultado es el siguiente:

```
[  
  {  
    "id": "medMG2",  
    "type": "microGenerator",  
    "category": {  
      "type": "Text",  
      "value": "solar",  
      "metadata": {}  
    },  
    "location": {  
      "type": "geo:point",  
      "value": "19.35307, -99.16304",  
      "bbox": "190 723 494 914"  
    }  
  }]
```

```
        "metadata": {}
    },
},
{
    "id": "medMG3",
    "type": "microGenerator",
    "category": {
        "type": "Text",
        "value": "solar",
        "metadata": {}
    },
    "location": {
        "type": "geo:point",
        "value": "19.35093, -99.16384",
        "metadata": {}
    }
},
{
    "id": "medMG7",
    "type": "microGenerator",
    "category": {
        "type": "Text",
        "value": "eolic",
        "metadata": {}
    },
    "location": {
        "type": "geo:point",
        "value": "19.35279, -99.16579",
        "metadata": {}
    }
}
]
```



5.- Lector de temperatura y humedad con Arduino

Introducción

Arduino es probablemente la plataforma de hardware abierto más popular para desarrollo y prototipado rápido de proyectos de sistemas digitales. Ha logrado conformar una amplísima comunidad de desarrolladores que comparten las especificaciones y el código de sus proyectos.

La plataforma tiene una serie de interfaces entrada y salida (E/S) que pueden ser programadas para monitorear (con ayuda de sensores) o interactuar (con ayuda de actuadores) con su entorno. La Figura 1 muestra la tarjeta Arduino YÚN, que es la que se utilizará en estas prácticas. La gran ventaja de esta tarjeta es que ya tiene integrada una interfaz WiFi.

Arduino cuenta con un ambiente de desarrollo integrado (IDE, Integrated Development Environment) para programar las acciones que se desean configurar en la placa. Los programas (llamados sketch) escritos en el IDE se descargan a la placa a través de la interfaz USB.

En esta sección nos familiarizaremos con el ambiente de desarrollo escribiendo programas sencillos para interactuar con la placa Arduino y, a través de ella, con algunos sensores y actuadores.

Hay una gran variedad de sensores (por ejemplo, de temperatura, humedad, iluminación, sonido, contaminantes, nutrientes, posicionamiento, velocidad, ...) y actuadores (por ejemplo, LEDs, zumbadores, relevadores, acopladores, ...) que pueden conectarse a una placa Arduino a través de sus interfaces programables E/S.

En principio, queremos implementar una red de sensores que sean capaces de detectar condiciones de riesgo en el medio ambiente (presencia de bacterias, gases o algún contaminante) en un hospital.

Por supuesto, implementar esta red en condiciones reales es sumamente peligroso, por lo que nos limitaremos a trabajar con un sensor de temperatura para crear nuestra primera prueba de concepto. A partir de ella, sería más sencillo diseñar la red de sensores hospitalaria. Bastará con identificar los sensores de interés, configurarlos y sustituir nuestros sensores de temperatura por aquéllos.

Materiales • Una placa Arduino YÚN y cable USB para conectarla a la PC • El software Arduino IDE • Una tarjeta protoboard y cables • Una resistencia de 4.7 kOhm • Sensor de temperatura y humedad DHT11

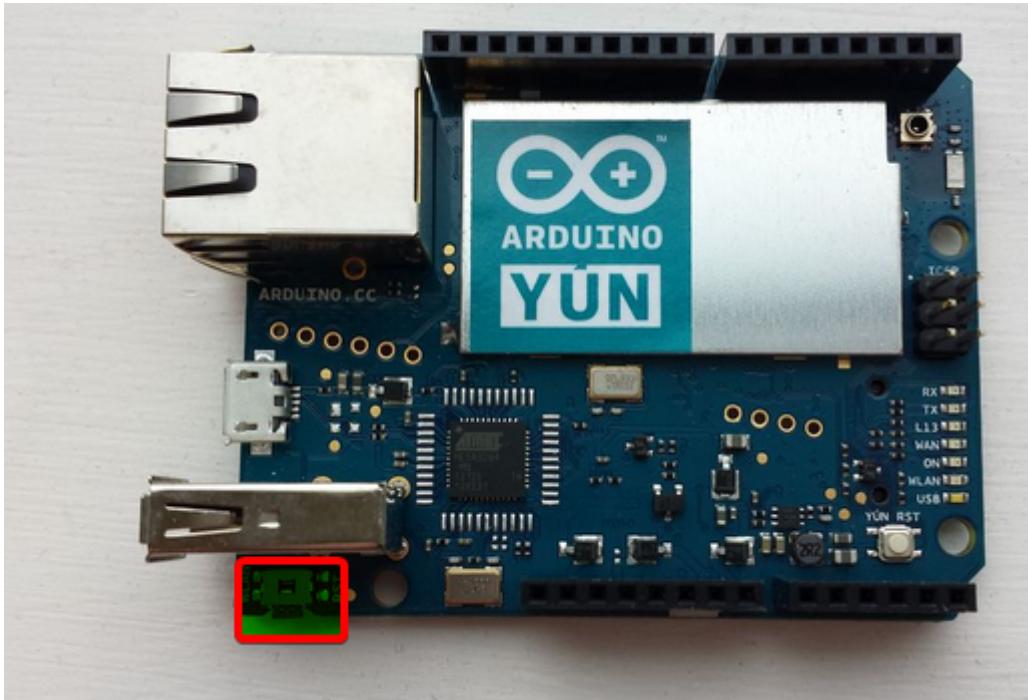
5.1. Instalación del IDE y detección de la placa Arduino

El IDE permite expresar un programa en un lenguaje muy parecido a C que se compila en el lenguaje de máquina del micro-controlador de la placa. Descargue el IDE de la siguiente dirección:

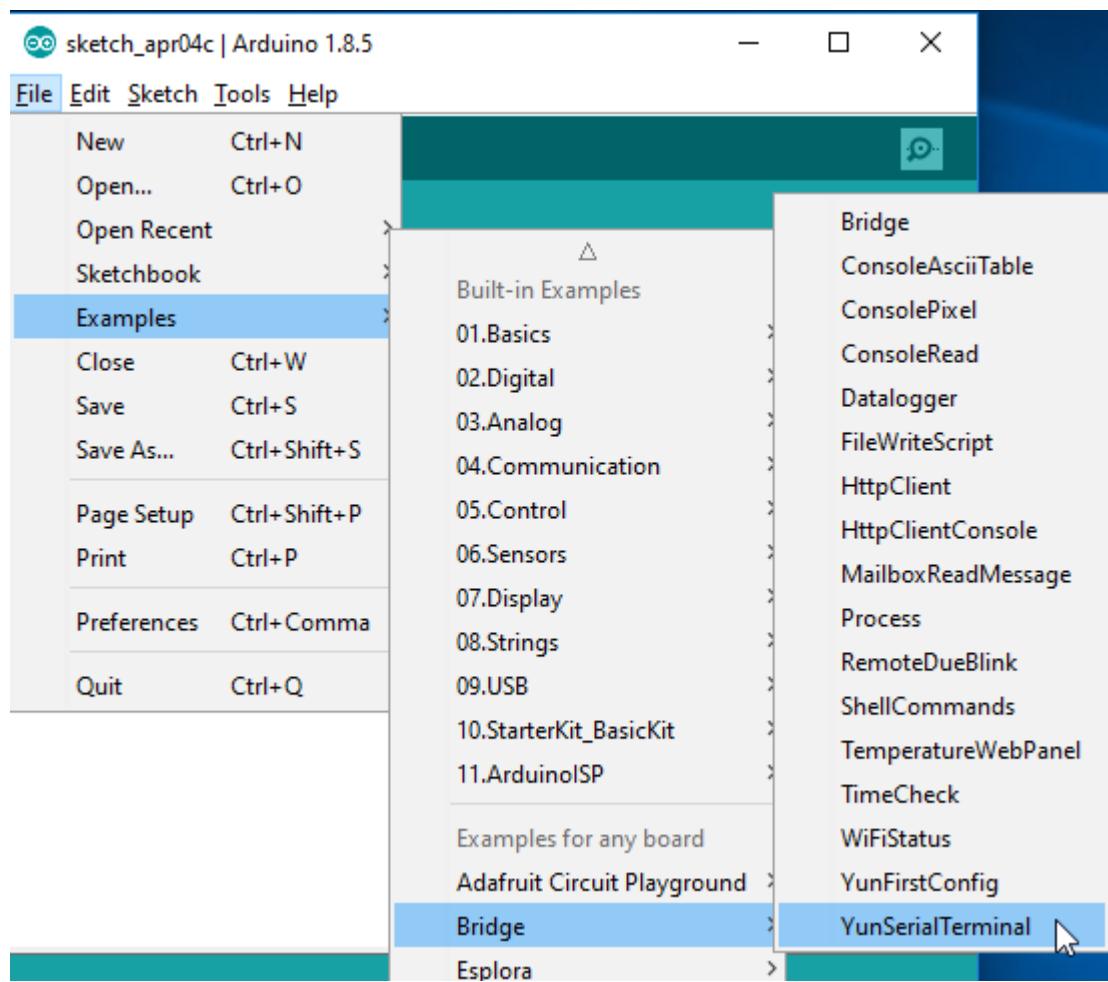
<https://www.arduino.cc/en/Main/Software>

Conecte la placa a un puerto USB de su computadora. Se deberá encender un LED verde indicando que la placa está energizada (aproximadamente un min).

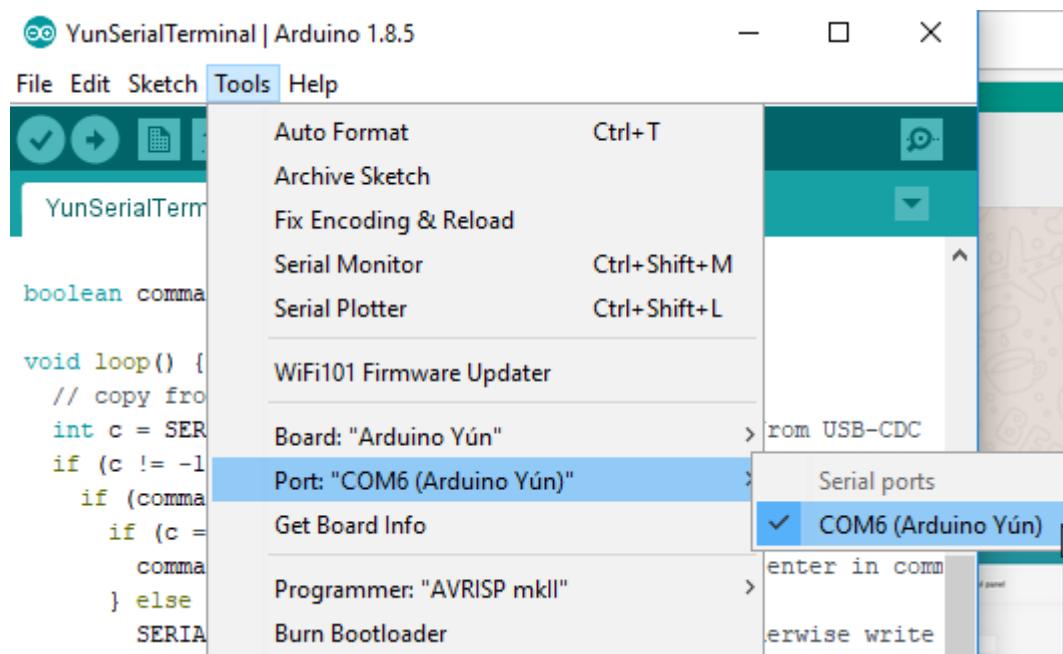
- Presionar el botón de Reset WLAN y mantener presionado por al menos 35 segundos.



- Esperar un minuto.
En el IDE de Arduino:
 - Ir a File -> Examples -> Bridge -> Yun Serial Terminal.



- Posteriormente, ir a Tools -> Port y seleccionar el puerto en donde esté el Arduino Yun.



- Dar clic en la flecha que se encuentra arriba al lado izquierdo para compilar y transferir este sketch a la tarjeta. Esperar a que se vea el mensaje *Done uploading* y de clic en la lupa que se encuentra arriba del lado derecho para abrir el monitor serie.

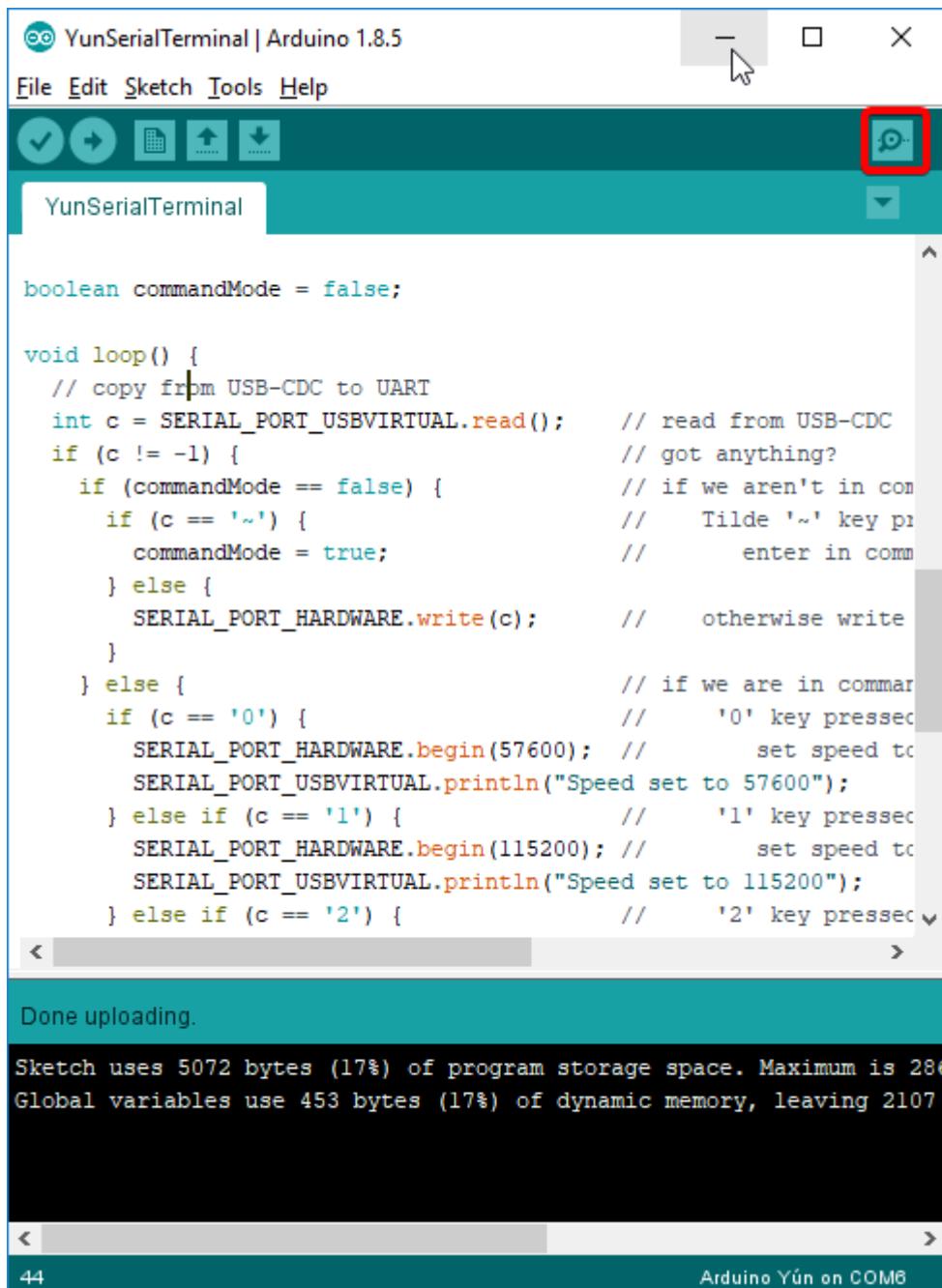
```
YunSerialTerminal | Arduino 1.8.5
File Edit Sketch Tools Help
YunSerialTerminal

boolean commandMode = false;

void loop() {
    // copy from USB-CDC to UART
    int c = SERIAL_PORT_USBVIRTUAL.read();      // read from USB-CDC
    if (c != -1) {                                // got anything?
        if (commandMode == false) {                // if we aren't in com
            if (c == '~') {                        //     Tilde '~' key pr
                commandMode = true;                 //         enter in comm
            } else {
                SERIAL_PORT_HARDWARE.write(c);      //         otherwise write
            }
        } else {                                 // if we are in commar
            if (c == '0') {                      //     '0' key pressed
                SERIAL_PORT_HARDWARE.begin(57600); //         set speed to
                SERIAL_PORT_USBVIRTUAL.println("Speed set to 57600");
            } else if (c == '1') {                //     '1' key pressed
                SERIAL_PORT_HARDWARE.begin(115200); //         set speed to
                SERIAL_PORT_USBVIRTUAL.println("Speed set to 115200");
            }
        }
    }
}

Done uploading.

Arduino Yún on COM6
```



```
boolean commandMode = false;

void loop() {
    // copy from USB-CDC to UART
    int c = SERIAL_PORT_USBVIRTUAL.read();      // read from USB-CDC
    if (c != -1) {                                // got anything?
        if (commandMode == false) {                // if we aren't in com
            if (c == '~') {                        //     Tilde '~' key pr
                commandMode = true;                 //         enter in comm
            } else {
                SERIAL_PORT_HARDWARE.write(c);      //         otherwise write
            }
        } else {                                    // if we are in commar
            if (c == '0') {                         //     '0' key pressed
                SERIAL_PORT_HARDWARE.begin(57600); //         set speed to
                SERIAL_PORT_USBVIRTUAL.println("Speed set to 57600");
            } else if (c == '1') {                  //     '1' key pressed
                SERIAL_PORT_HARDWARE.begin(115200); //         set speed to
                SERIAL_PORT_USBVIRTUAL.println("Speed set to 115200");
            } else if (c == '2') {                  //     '2' key pressed

```

Done uploading.

Sketch uses 5072 bytes (17%) of program storage space. Maximum is 286 Global variables use 453 bytes (17%) of dynamic memory, leaving 2107

- En la ventana que se muestra seleccionar en el primer menú inferior “NewLine” y en el segundo menú “115200 baud”

```
ifconfig
```

root@Arduino01:#
root@Arduino01:#
root@Arduino01:#

Autoscroll

- En el espacio superior escribir `ifconfig` y dar enter hasta que se muestre la dirección ip en el puerto eth1.

```
COM6
| Send
root@Arduino01:#  
root@Arduino01:#  
root@Arduino01:#  
root@Arduino01:#  
root@Arduino01:# ifconfig  
eth1      Link encap:Ethernet HWaddr 90:A2:DA:FD:04:22  
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1  
          RX packets:8 errors:0 dropped:1 overruns:0 frame:0  
          TX packets:4 errors:0 dropped:0 overruns:0 carrier:0  
          collisions:0 txqueuelen:1000  
          RX bytes:480 (480.0 B)  TX bytes:1572 (1.5 KiB)  
          Interrupt:4  
  
lo       Link encap:Local Loopback  
          inet addr:127.0.0.1 Mask:255.0.0.0  
          UP LOOPBACK RUNNING MTU:65536 Metric:1  
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0  
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0  
          collisions:0 txqueuelen:0  
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)  
  
wlan0    Link encap:Ethernet HWaddr 90:A2:DA:F5:04:22  
          inet addr:192.168.240.1 Bcast:192.168.240.255 Mask:255.255.255.0  
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1  
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0  
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0  
          collisions:0 txqueuelen:32  
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

```
root@Arduino01:#  
< Newline 115200 baud Clear output  
 Autoscroll  
COM6
| Send
root@Arduino01:#  
root@Arduino01:#  
root@Arduino01:# ifconfig  
eth1      Link encap:Ethernet HWaddr 90:A2:DA:FD:04:22  
          inet addr:177.245.225.6 Bcast:177.245.225.63 Mask:255.255.255.192  
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1  
          RX packets:3604 errors:0 dropped:173 overruns:0 frame:0  
          TX packets:635 errors:0 dropped:0 overruns:0 carrier:0  
          collisions:0 txqueuelen:1000  
          RX bytes:401886 (392.4 KiB)  TX bytes:110494 (107.9 KiB)  
          Interrupt:4  
  
lo       Link encap:Local Loopback  
          inet addr:127.0.0.1 Mask:255.0.0.0  
          UP LOOPBACK RUNNING MTU:65536 Metric:1  
          RX packets:112 errors:0 dropped:0 overruns:0 frame:0  
          TX packets:112 errors:0 dropped:0 overruns:0 carrier:0  
          collisions:0 txqueuelen:0  
          RX bytes:8093 (7.9 KiB)  TX bytes:8093 (7.9 KiB)  
  
wlan0    Link encap:Ethernet HWaddr 90:A2:DA:F5:04:22  
          inet addr:192.168.240.1 Bcast:192.168.240.255 Mask:255.255.255.0  
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1  
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0  
          TX packets:15 errors:0 dropped:0 overruns:0 carrier:0
```

```

    collisions:0 txqueuelen:32
    RX bytes:0 (0.0 B)  TX bytes:3065 (2.9 KiB)

root@Arduino01:/#

```

Autoscroll

Newline

115200 baud

Clear output

- Con el comando `ifconfig` obtenemos la dirección IP que tiene asignada el Arduino. Anote esta IP porque la utilizará más adelante. En nuestro caso, la dirección es la **177.245.225.6**.
-
- Para comprobar que el Arduino está conectado a Internet ponga en el espacio superior `ping 8.8.8.8 -c 3`. Se debe mostrar el mensaje que se encuentra en la parte inferior de la siguiente imagen:

The screenshot shows a terminal window titled "COM6" with the following content:

```

ping 8.8.8.8 -c 3
collisions:0 txqueuelen:1000
RX bytes:76516 (74.7 KiB)  TX bytes:17329 (16.9 KiB)
Interrupt:4

lo      Link encap:Local Loopback
        inet addr:127.0.0.1  Mask:255.0.0.0
        UP LOOPBACK RUNNING MTU:65536 Metric:1
        RX packets:64 errors:0 dropped:0 overruns:0 frame:0
        TX packets:64 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
        RX bytes:5306 (5.1 KiB)  TX bytes:5306 (5.1 KiB)

wlan0    Link encap:Ethernet HWaddr 90:A2:DA:F5:04:22
        inet addr:192.168.240.1  Bcast:192.168.240.255  Mask:255.255.255.0
        UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
        RX packets:0 errors:0 dropped:0 overruns:0 frame:0
        TX packets:15 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:32
        RX bytes:0 (0.0 B)  TX bytes:3065 (2.9 KiB)

root@Arduino01:/# ping 8.8.8.8 -c 3
PING 8.8.8.8 (8.8.8.8): 56 data bytes
64 bytes from 8.8.8.8: seq=0 ttl=51 time=30.369 ms
64 bytes from 8.8.8.8: seq=1 ttl=51 time=30.302 ms
64 bytes from 8.8.8.8: seq=2 ttl=51 time=30.305 ms

--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 30.302/30.325/30.369 ms
root@Arduino01:/#

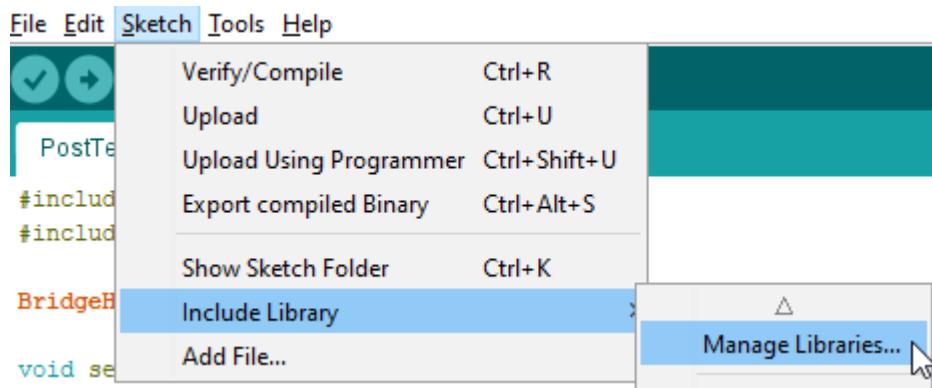
```

At the bottom of the terminal window, there are three buttons: "Autoscroll" (checked), "Newline", and "115200 baud".

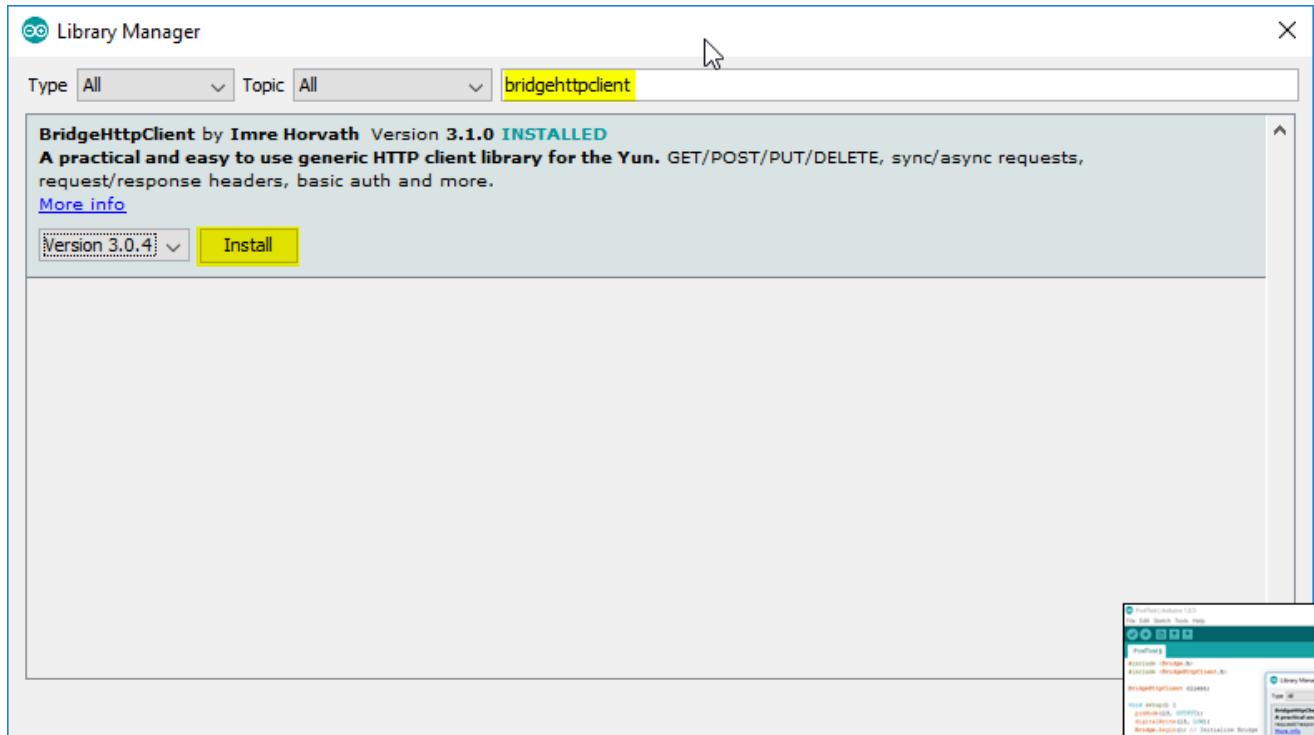
Enviar datos desde el Arduino hacia el OCB

Para esta sección debemos crear un nuevo sketch vacío y seguir los siguientes pasos:

** 1.-** Instalar la librería `BridgeHttpClient` dando clic en *Sketch -> Include Library -> Manage Libraries*



- Buscar la librería y dar clic en el botón "Install".



2.- Posteriormente pegar el siguiente código:

```
#include <Bridge.h>
#include <BridgeHttpClient.h>

BridgeHttpClient client;

void setup() {
  pinMode(13, OUTPUT);
  digitalWrite(13, LOW);
  Bridge.begin(); // Initialize Bridge
  digitalWrite(13, HIGH);

  SerialUSB.begin(115200);
  while (!SerialUSB); // wait for a serial connection
  client.addHeader("Content-Type: application/json");
  SerialUSB.println("Finish setup");
}
```

```

void loop() {
    SerialUSB.println("Sending...");

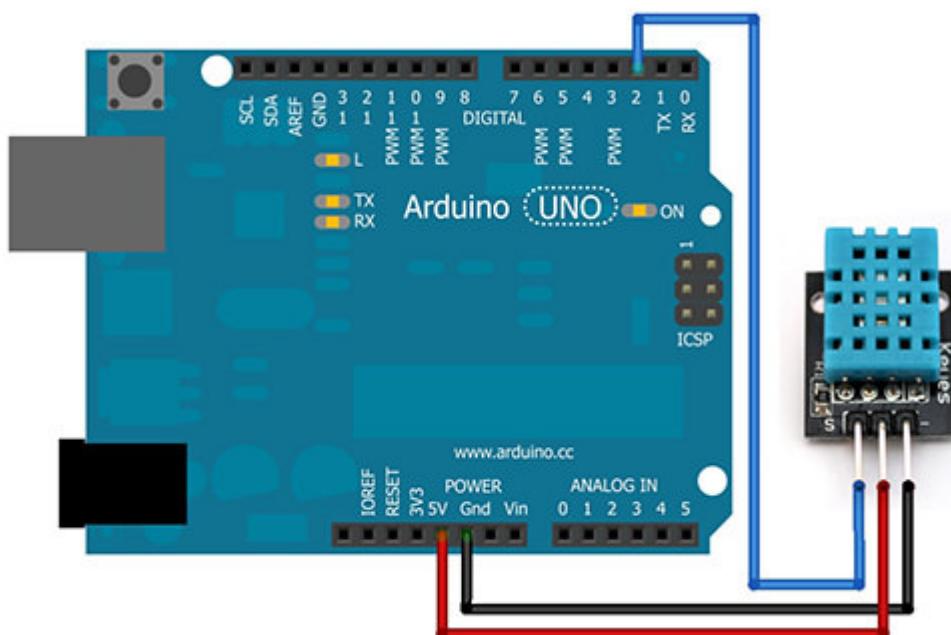
    client.enableInsecure(); // Using HTTPS and peer cert. will not be able to auth.
    client.post("http://130.206.112.201:1026/v2/entities",
    {"id": "RoomArduinoXXY"});
    SerialUSB.println("Termino");
    while(1);
}

```

3.- Cargarlo al Arduino y abrir el monitor serial. Para comprobar que haya funcionado se debe de obtener algo parecido a lo que se muestra en la siguiente imagen:



4.- Es momento de conectar el sensor de temperatura y humedad **DHT11** a la tarjeta Arduino como se muestra en la siguiente figura. **NOTAS:** - Aunque la imagen muestra un Arduino UNO, el esquema de conexiones es idéntico para el YÚN - El circuito que se muestra ya tiene una resistencia de 4.7 kOhm entre la pata de Voltaje y la de señal. - La pata de voltaje se conecta al pin 5V, la de tierra a Gnd y la de señal al pin 2 de E/S



5.- Ahora vamos a instalar dos nuevas librerías llamadas **dht sensor library** y **adafruit unified sensor by adafruit**.

6.- Una vez instaladas, creamos un nuevo sketch y pegamos el siguiente código:

```
#include <DHT.h>
```

```
// Definimos el pin digital donde se conecta el sensor
#define DHTPIN 2
// Dependiendo del tipo de sensor
#define DHTTYPE DHT11

// Inicializamos el sensor DHT11
DHT dht(DHTPIN, DHTTYPE);

void setup() {
    // Inicializamos comunicación serie
    serial.begin(9600);

    // Comenzamos el sensor DHT
    dht.begin();
}

void loop() {
    // Esperamos 5 segundos entre medidas
    delay(5000);

    // Leemos la humedad relativa
    float h = dht.readHumidity();
    // Leemos la temperatura en grados centígrados (por defecto)
    float t = dht.readTemperature();
    // Leemos la temperatura en grados Fahrenheit
    float f = dht.readTemperature(true);

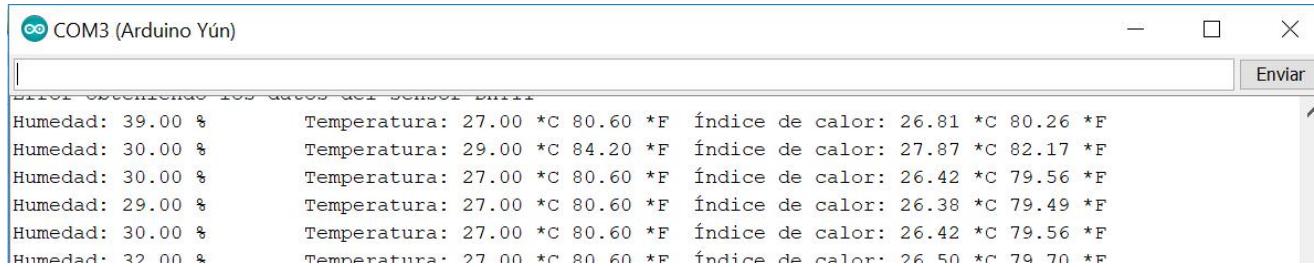
    // Comprobamos si ha habido algún error en la lectura
    if (isnan(h) || isnan(t) || isnan(f)) {
        Serial.println("Error obteniendo los datos del sensor DHT11");
        return;
    }

    // Calcular el índice de calor en Fahrenheit
    float hif = dht.computeHeatIndex(f, h);
    // Calcular el índice de calor en grados centígrados
    float hic = dht.computeHeatIndex(t, h, false);

    serial.print("Humedad: ");
    serial.print(h);
    serial.print(" %\t");
    serial.print("Temperatura: ");
    serial.print(t);
    serial.print(" *C ");
    serial.print(f);
    serial.print(" *F\t");
    serial.print("Índice de calor: ");
    serial.print(hic);
    serial.print(" *C ");
    serial.print(hif);
    serial.println(" *F");
}
```

}

En el monitor serie puede comprobar que el Arduino está tomando las lecturas y las está enviando al monitor como se muestra en esta figura:



```
Humedad: 39.00 %
Temperatura: 27.00 *C 80.60 *F índice de calor: 26.81 *C 80.26 *F
Humedad: 30.00 %
Temperatura: 29.00 *C 84.20 *F índice de calor: 27.87 *C 82.17 *F
Humedad: 30.00 %
Temperatura: 27.00 *C 80.60 *F índice de calor: 26.42 *C 79.56 *F
Humedad: 29.00 %
Temperatura: 27.00 *C 80.60 *F índice de calor: 26.38 *C 79.49 *F
Humedad: 30.00 %
Temperatura: 27.00 *C 80.60 *F índice de calor: 26.42 *C 79.56 *F
Humedad: 32.00 %
Temperatura: 27.00 *C 80.60 *F índice de calor: 26.50 *C 79.70 *F
```

¡Ahora es su turno!

Su siguiente tarea es integrar ambas funcionalidades para lograr que el Arduino mande los datos que está leyendo de humedad y temperatura al OCB.

Para esto tendrá que:

- Definir el modelo de datos.
- Crear el JSON correspondiente a dicho modelo.
- Crear una entidad en el OCB correspondiente a su modelo.
- Actualizar los datos desde el Arduino cada segundo.

5.2..- Visualizar los datos capturados en un tablero de control.

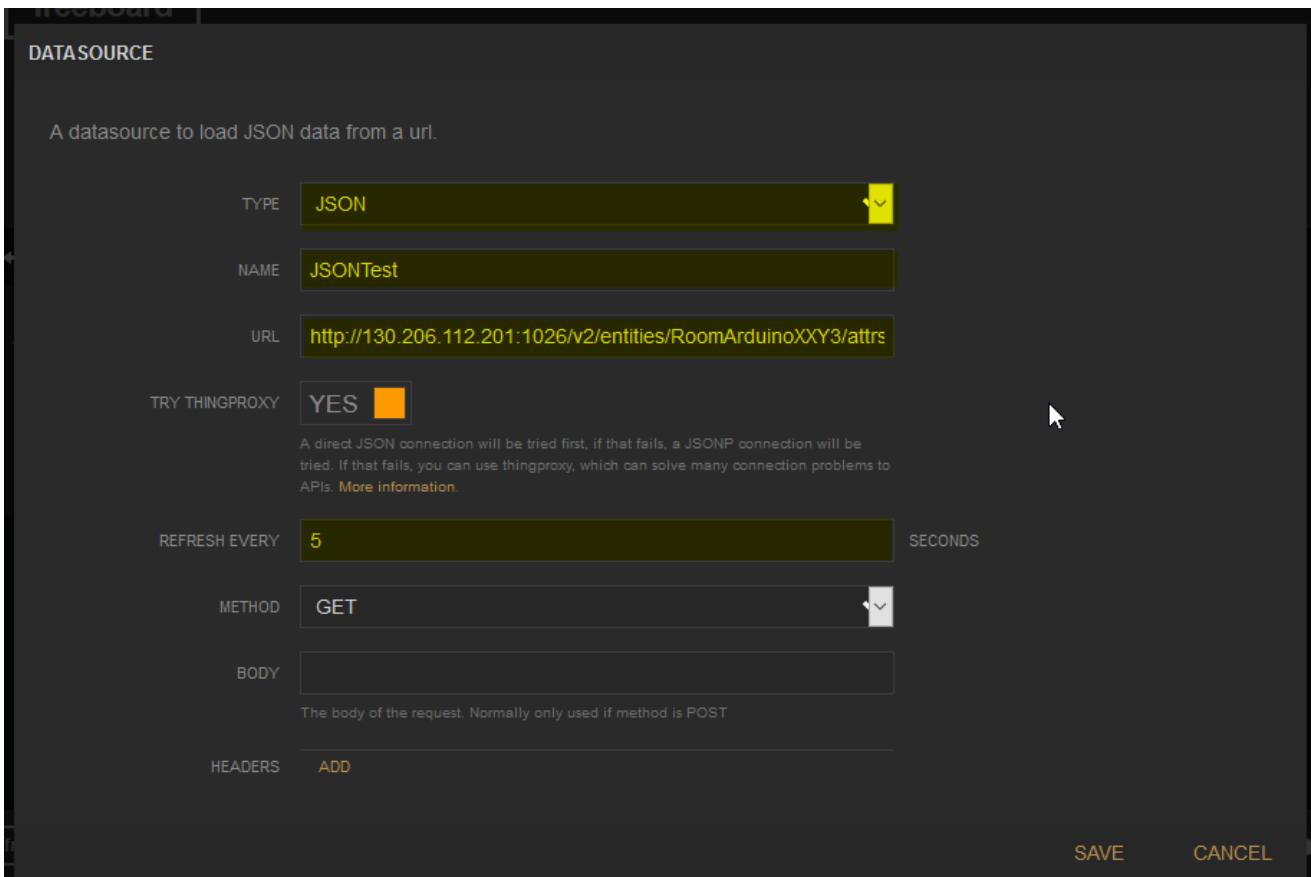
Para poder visualizar los datos capturados con la ayuda del Arduino vamos a utilizar una herramienta llamada **Freeboard**. Para hacerlo seguiremos los siguientes pasos:

1.- Ir a la página www.freeboard.io, dar clic en start now y registrarse.

2.- Una vez registrado dar clic en el botón *create new*.

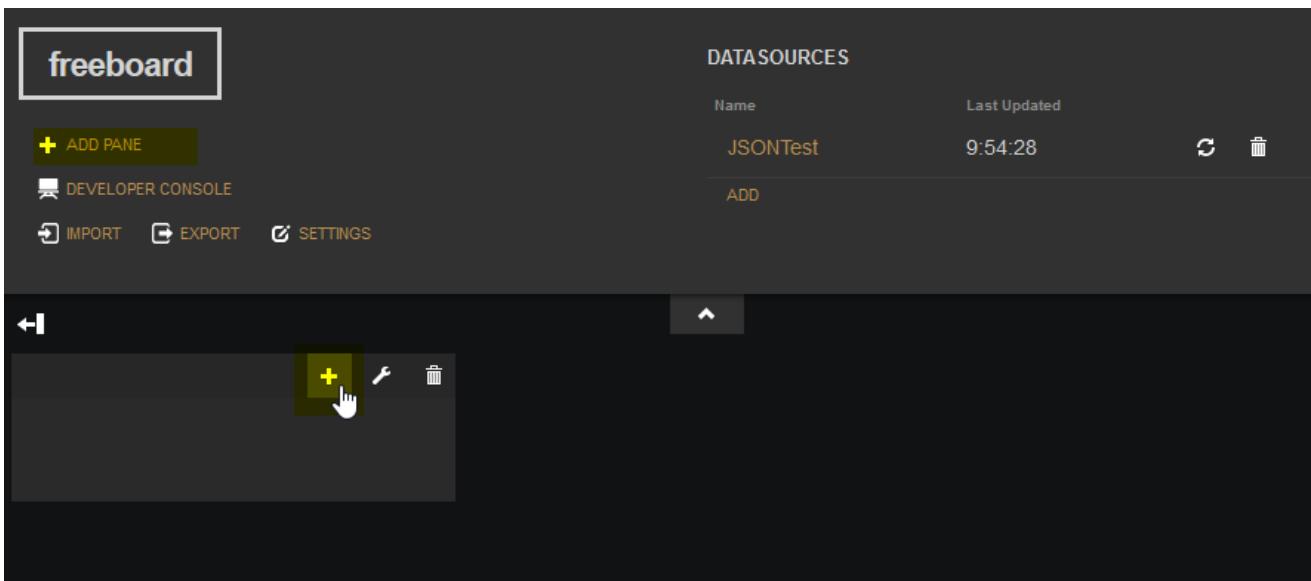
3.- En la forma se tiene que elegir **Add** en **DATASOURCES** con la siguiente información:

- Type -> JSON
- Name -> cualquiera que le haga sentido
- URL -> la dirección en la cual sabemos que el OCB va a responder **con el valor del atributo deseado**
- Refresh every -> 5 segundos.



Por simplicidad hemos elegido un mecanismo de actualización **por poleo**. Nuestro panel lanzará una consulta cada 5 segundos al OCB para actualizar los valores de temperatura y humedad. Una forma alterna, frecuentemente preferida, es que el OCB le notifique al panel cada vez que recibe un nuevo valor. Para implementar esta alternativa, llamada **por subscripción**, es necesario registrar el panel en Freeboard como una entidad a la que se le deben notificar las actualizaciones.

4.- Posteriormente dar clic en ``add pane y en el símbolo de mas que aparece abajo.



- En la forma que aparece seleccionar el tipo de tablero deseado, darle un título, en el valor poner "datasource[nombredeDataSource]["value"]" y en las unidades poner °C y guardar.

WIDGET

The screenshot shows the configuration interface for a 'Text' widget. The configuration fields are as follows:

- TYPE:** Text
- TITLE:** Temperatura
- SIZE:** Regular
- VALUE:** datasources["JSONTest"]["value"]
- INCLUDE SPARKLINE:** NO
- ANIMATE VALUE CHANGES:** YES
- UNITS:** ° C

At the bottom right, there are 'SAVE' and 'CANCEL' buttons.

¡Ahora es tu turno!

Tu siguiente tarea es integrar ambas funcionalidades para lograr que el Arduino mande los datos que está leyendo de humedad y temperatura al OCB.

Para esto tendrás que:

- Definir el modelo de datos.
- Crear el JSON correspondiente a dicho modelo.
- Crear una entidad en el OCB correspondiente a tu modelo.
- Actualizar los datos desde el Arduino cada segundo.