

Tutorial Fiware - Orion Context Broker

Internet de las cosas en la salud

Taller Talent Land, Abr 2018

Introducción

En la presentación que se ha hecho como introducción al taller, 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.

Hemos puesto la atención en las muchas aplicaciones que se han desplegado en el sector salud, desde las pulseras y bandas para medir signos vitales, hasta monitores de partículas peligrosas en zonas hospitalarias.

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.

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 en una primera fase, y volver a él con más calma al terminar el taller.

El taller está compuesto de cuatro bloques:

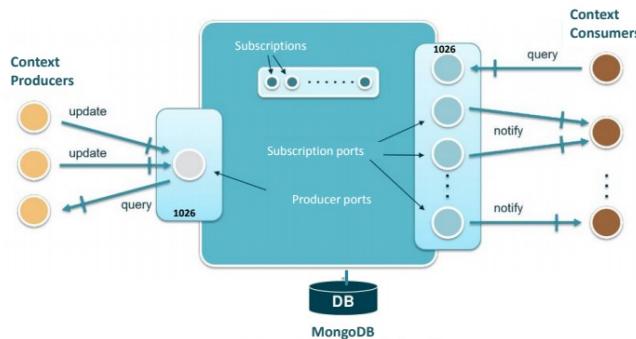
1. Familiarizarse con el estándar de representación de datos NGSI y con los principales comandos que maneja el Orion Context Broker.
 - Interactuar con el OCB con mensajes REST en formato JSON
2. Implementar el prototipo de una estación de monitoreo (temperatura y humedad) con una tarjeta Arduino
 - 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

1.- 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), 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 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; en nuestro caso, 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.

Representación de datos de contexto

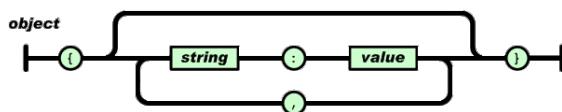
Para representar objetos de la vida real se utiliza el modelo de entidades de la API NGSI. En éste se define un **modelo de datos** de información de contexto basado en **entidades** y en **atributos**. Cada entidad representa un objeto de la vida real y puede tener atributos y metadatos.

Las entidades cuentan con un identificador (ID) y un tipo. Esta parte ID/tipo debe

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 podrían 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 tienen la siguiente forma:



Es decir, se encuentran definidos entre llaves. String será definido como las propiedades entidades. Los value son los atributos.

Por ejemplo, modelaremos la temperatura y la presión de un cuarto con la siguiente entidad:

```
{
  "id": "Cuarto1",
  "type": "Cuarto",
  "temperature": {
    "type": "Float",
    "value": 23,
    "metadata": {
      "precision": {
        "type": "xxx",
        "value": xxx
      }
    }
  },
  "pressure": {
    "value": 720
  }
}
```

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.

Interactuando con el OCB

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.

Este tipo de servicios permiten obtener información de forma arborecente. Es decir, es posible obtener | actualizar | borrar información de una entidad completa o sólo valores de una entidad en específico.

El URL al que haremos la solicitud sera: <http://XX.XX.XX.XX:1026/v2/>..., es decir, la comunicación se hace a través del puerto 1026 y utilizando la versión 2 de la NGSI.

(Sustituya las XX.XX.XX.XX por la dirección IP de su instancia si la instaló como se indica en la cuarta parte de este tutorial, o la dirección IP del servidor que se le indique en la sala.)

Los métodos REST que utilizaremos son **GET, POST, PUT, DELETE, OPTIONS, HEAD, TRACE, CONNECT**.

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**.

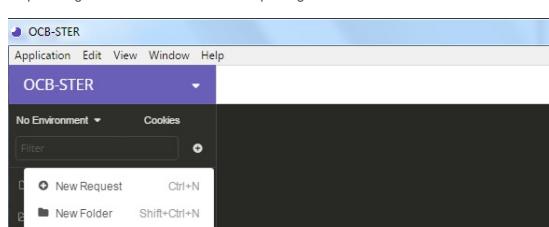
Para indicar que se está enviando información se debe de poner **Content-Type** y para indicar que se desea recibir se debe de poner **Accept**.

Así, una solicitud quedaría de la siguiente manera:

```
xx.xx.xx:pto/v2/entities
Método: POST
Headers: Content-Type:application/json
Body:
{ "id": xx
  "type": xx
  "atributo":{
    "value":xx
  }
...
}
```

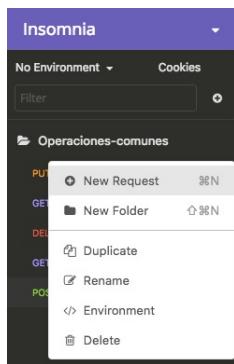
Para poder interactuar con el OCB utilizaremos la herramienta **Insomnia**. Si lo desea, puede utilizar cualquier otro cliente REST. De hecho, se puede hacer desde la terminal de git con el comando **curl**, pero ello es mucho más propenso a errores.

Crearemos en insomnia una carpeta llamada **Operaciones-Comunes**. En esta carpeta se guardarán todas las consultas que hagamos.



POST

En primer lugar debemos enviar la representación de una entidad con el método POST: Comenzaremos por crear una nueva petición (New Request) en Insomnia:



El nombre sugerido para esta petición es **inserta-entidad**, el método que utilizaremos será **POST** y el cuerpo (body) será de tipo JSON.

El URL que utilizaremos será <http://XX.XX.XX.XX:1026/v2/entities> y el tipo de encabezado será *application/json*. En Insomnia se establece automáticamente cuando seleccionamos JSON como el tipo de dato (en algunas operaciones posteriores, utilizaremos un body tipo *text/plain*).

```
{  
    "id": "Room01",  
    "type": "Room",  
    "floor" :{  
        "value":"PB",  
        "type":"text"  
    },  
    "temperature" :{  
        "type": "Float",  
        "value": 23  
    }  
}
```

En el cuerpo de la consulta, en la parte media de Insomnia, colocaremos la descripción de la entidad

A screenshot of the Insomnia application interface titled 'OCB-STER - inserta-entidad'. The request details show a POST method to '127.0.0.1:1026/v2/entities'. The response status is '201 CREATED' with a 'TIME 0 ms' and 'SIZE' of 0. The 'Preview' tab shows the JSON response body: 'No body returned for response'. The left sidebar shows various operations like 'verifica ocb', 'Consulta entidades', etc.

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.

EJERCICIO. Agregue otras dos entidades con los siguientes valores:

- Como identificador, Roomzz1, Roomzz2 (**sustituya zz1 y zz2 por los caracteres que desee -por ejemplo, sus iniciales- asegurando que ningún otro equipo asigne el mismo identificador**).
- Temp 22 y 27.3

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>:

A screenshot of the Insomnia application interface titled 'OCB-STER - obten-todas-entidades'. The request details show a GET method to 'http://127.0.0.1:1026/v2/entities'. The response status is '200 OK' with a 'TIME 140 ms' and 'SIZE 655 B'. The 'Preview' tab shows the JSON response body containing multiple entity definitions. The left sidebar shows the 'obten-todas-entidades' operation.

Consulta acotada.

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

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

```

1: {
2:   "id": "Med025",
3:   "type": "LineMeter",
4:   "current": {
5:     "type": "float",
6:     "value": 2.305,
7:     "metadata": {}
8:   },
9:   "frequency": {
10:     "type": "Number",
11:     "value": 1.9987,
12:     "metadata": {}
13:   }
14: }

```

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

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

Por ejemplo, para ver el atributo "temperature" de la entidad Room01, se utiliza el URL <http://XX.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.XX:1026/v2/entities/Room01/attrs/temperature/value/>.

EJERCICIO. Agregue una nueva consulta en Insomnia. Nombrela consulta-por-atributos y modifíquela para consultar el atributo temperature de las dos entidades que usted creó.

Los resultados obtenidos deben ser los siguientes:

```
{
  "type": "Float",
  "value": 22,
  "metadata": {}
}

y

27.3
```

Actualización de valores

Si deseamos actualizar los valores de 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 *frequency* y *current* de la entidad *Med024*:

Método: PUT
{url}/v2/entities/{id}/attrs/{value}

Así tenemos:

```
http://XX.XX.XX.XX:1026/v2/entities/Room01/attrs/temperature
Método: PUT
Headers: Content-Type:application/json
Body;

{
  "type": "Float",
  "value": 19
}
```

Insomnia – UpdateCompleto

Para verificar el cambio se puede volver a hacer el método GET en donde veremos que el valor del atributo ha cambiado.

Insomnia – Pedir
Application Edit View Window Tools Help

```

1: [
2:   {
3:     "id": "ROOM01",
4:     "type": "Room",
5:     "floor": {
6:       "type": "text",
7:       "value": "PB",
8:       "metadata": {}
9:     },
10:    "temperature": {
11:      "type": "float",
12:      "value": 100,
13:      "metadata": {}
14:    }
15:  }
16: ]

```

Con este método, si se omite un atributo, éste desaparece de la entidad. 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* Room01, la consulta se hará así:

Método: PATCH
URL: <http://XX.XX.XX.XX:1026/v2/entities/Room01/attrs>

```

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

Header: Content-type: application/json

```

Frecuentemente, lo que se desea es actualizar únicamente el valor de un atributo. En este caso, como se hizo anteriormente, se extiende el URL hasta {attrId}/value y en el cuerpo del mensaje se coloca el valor, especificando que el tipo de contenido es texto plano.

Delete

El método DELETE permite eliminar entidades y atributos.

Para borrar un atributo se utiliza el comando Delete:

```
delete http://url:1026/v2/entities/{id}/attrs/{value}
```

Para borrar una se utiliza la siguiente expresión:

```
delete http://url:1026/v2/entities/{id}
```

Para probar este método, creamos una nueva entidad:

```
{
    "id": "talentland",
    "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:1026/v2/entities/ineel/attrs/NumGente>.

Verifique que la variable fue eliminada.

Ahora elimine la entidad completa con la URL
<http://XX.XX.XX:1026/v2/entities/ineel>.

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; NGSIV2 los considerará "Número".

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

Por supuesto, sólo un equipo lo puede hacer con estos identificadores. Si lo desea, modifique el cuerpo asignando identificadores propios

```
{
    "actionType": "APPEND",
    "entities": [
        {
            "id": "Room02",
            "type": "Room",
            "floor": {
                "value": "PB"
            },
            "temperature": {
                "type": "Float",
                "value": 26
            }
        },
        {
            "id": "Room03",
            "type": "Room",
            "floor": {
                "value": "PA"
            },
            "temperature": {
                "type": "Float",
                "value": 30
            }
        },
        {
            "id": "Room04",
            "type": "Room",
            "floor": {
                "value": "PB"
            },
            "temperature": {
                "type": "Float",
                "value": 19
            }
        },
        {
            "id": "Room05",
            "type": "Room",
            "floor": {
                "value": "PA"
            },
            "temperature": {
                "type": "Float",
                "value": 16
            }
        }
    ]
}
```

```
{
    "id": "Room06",
    "type": "Room",
    "floor": {
        "value": "PB"
    },
    "temperature": {
        "type": "Float",
        "value": 12
    }
},
{
    "id": "Room07",
    "type": "Room",
    "floor": {
        "value": "PA"
    },
    "temperature": {
        "type": "Float",
        "value": 18
    }
},
{
    "id": "Room08",
    "type": "Room",
    "floor": {
        "value": "PA"
    },
    "temperature": {
        "type": "Float",
        "value": 25
    }
},
{
    "id": "Room09",
    "type": "Room",
    "floor": {
        "value": "PB"
    },
    "temperature": {
        "type": "Float",
        "value": 15
    }
},
{
    "id": "Room10",
    "type": "Room",
    "floor": {
        "value": "PB"
    },
    "temperature": {
        "type": "Float",
        "value": 101
    }
}
]
```

Header: Content-type: application/json

Verifique que las entidades se cargaron en el OCB.

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 cumplen 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` al query. 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:1026/v2/entities?limit=3&offset=5&type=LineMeter&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:

```

2: [
3:     {
4:         "id": "Med01",
5:         "type": "LineMeter",
6:         "current": 1.9681,
7:         "frequency": 59.7123,
8:         "voltage": 119.1956
9:     },
10:    {
11:        "id": "Med02",
12:        "type": "LineMeter",
13:        "current": 2.1615,
14:        "frequency": 59.9456,
15:        "voltage": 120.3477
16:    },
17:    {
18:        "id": "Med03",
19:        "type": "LineMeter",
20:        "current": 2.1935,
21:        "frequency": 59.6615,
22:        "voltage": 119.9467
23:    }
]

```

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 a 32.5 grados.

```
http://XX.XX.XX:1026/v2/entities?q=temperature>32.5&limit=20&options=keyValues
```

Con el parámetro `attrs` se pueden especificar qué atributos se desea desplegar.

EJERCICIO Muestre todas las entidades en las que la temperatura excede los 28 grados. Debe mostrar únicamente la temperatura.

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. (Solo 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": "19.34931, -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": {}
    }
}

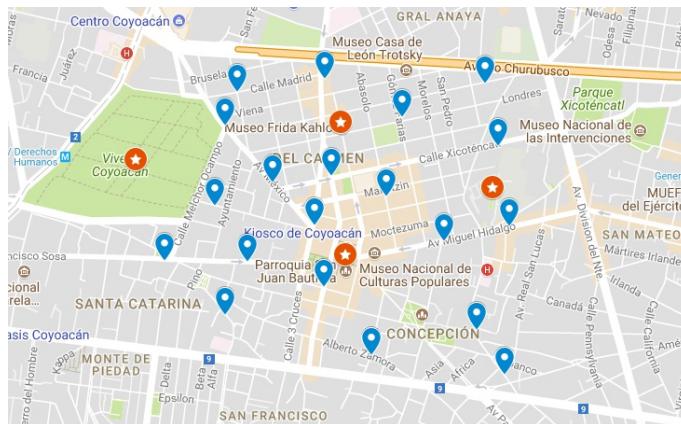
```

```

        "metadata": {}
    },
    "location": {
        "type": "geo:point",
        "value": "19.34702, -99.16796",
        "metadata": {}
    }
},
{
    "id": "medG17",
    "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 azú, junto con algunos puntos de interés en naranja.



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

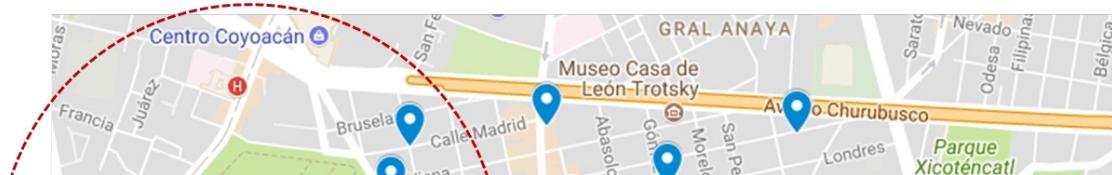
Por ejemplo, en la siguiente consulta se buscarán localidades microgeneradoras que estén como máximo a 800 metros de los viveros de Coyoacán representados por el punto 19.3538, -99.17208.

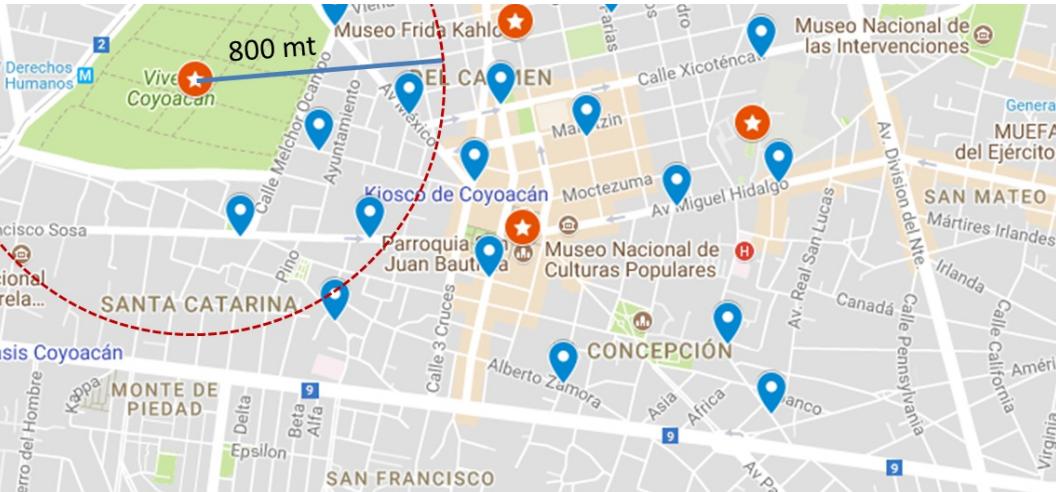
```

Método: GET
URL: http://XX.XX.XX: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": {}
    }
  },
  {
    "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

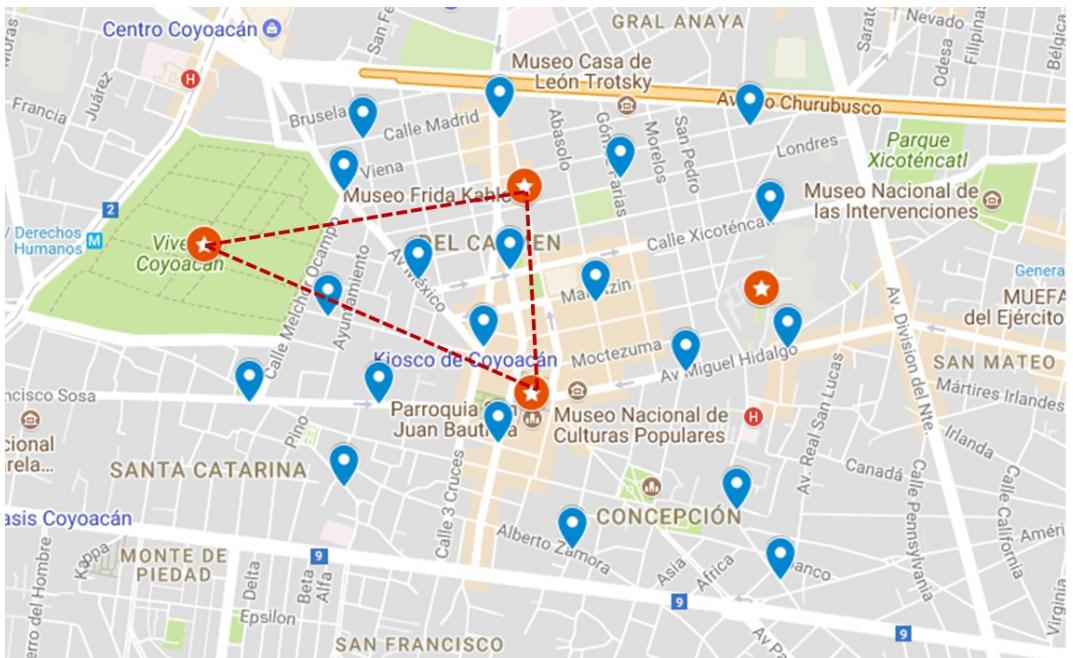
En el siguiente cuadro, haremos una geo-cerca. Buscaremos los microgeneradores

En el siguiente ejemplo, haremos una geoquery. Buscaremos los microgeneradores 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.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": "19.35387, -99.16304",  
      "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": {}  
    }  
  }  
]
```



2.- 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, será 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 YUN 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

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:

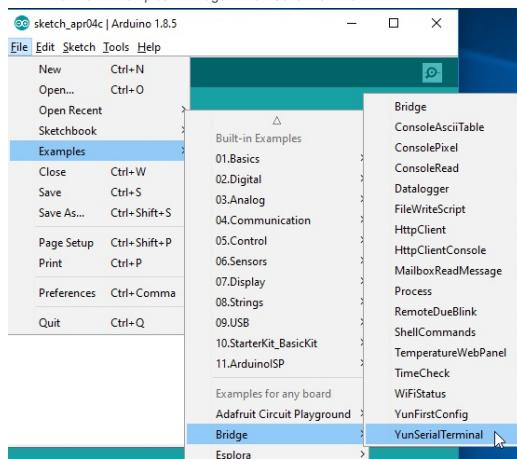
<https://www.arduino.cc/en/Main/Software>

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).

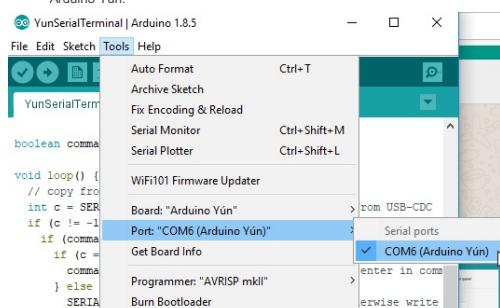
- Presionar el botón de Reset WLAN y mantener presionado por al menos 35 segundos.



- 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.



```

if (c != -1) {
    if (commandMode == false) { // got anything?
        // if we aren't in command mode
        if (c == '~') { // Tilde `~` key pressed
            commandMode = true; // enter in command mode
        } else {
            SERIAL_PORT_HARDWARE.write(c); // otherwise write
        }
    } else { // if we are in command mode
        if (c == '0') { // '0' key pressed
            SERIAL_PORT_HARDWARE.begin(57600); // set speed to 57600
            SERIAL_PORT_USBVIRTUAL.println("Speed set to 57600");
        } else if (c == '1') { // '1' key pressed
            SERIAL_PORT_HARDWARE.begin(115200); // set speed to 115200
            SERIAL_PORT_USBVIRTUAL.println("Speed set to 115200");
        } else if (c == '2') { // '2' key pressed
            SERIAL_PORT_HARDWARE.begin(250000); // set speed to 250000
            SERIAL_PORT_USBVIRTUAL.println("Speed set to 250000");
        }
    }
}

```

Done uploading

44 Arduino Yun on COM6

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 command mode
            if (c == '~') { // Tilde `~` key pressed
                commandMode = true; // enter in command mode
            } else {
                SERIAL_PORT_HARDWARE.write(c); // otherwise write
            }
        } else { // if we are in command mode
            if (c == '0') { // '0' key pressed
                SERIAL_PORT_HARDWARE.begin(57600); // set speed to 57600
                SERIAL_PORT_USBVIRTUAL.println("Speed set to 57600");
            } else if (c == '1') { // '1' key pressed
                SERIAL_PORT_HARDWARE.begin(115200); // set speed to 115200
                SERIAL_PORT_USBVIRTUAL.println("Speed set to 115200");
            } else if (c == '2') { // '2' key pressed
                SERIAL_PORT_HARDWARE.begin(250000); // set speed to 250000
                SERIAL_PORT_USBVIRTUAL.println("Speed set to 250000");
            }
        }
    }
}

```

Done uploading.

Sketch uses 5072 bytes (17%) of program storage space. Maximum is 28672 bytes. Global variables use 453 bytes (17%) of dynamic memory, leaving 2107 bytes available for local variables.

44 Arduino Yun on COM6

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

COM6

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

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:0 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:/#

Autoscroll

COM6

root@Arduino01:/#

```

root@arduino01:~#
root@arduino01:~# ifconfig
eth0      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

- 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:

```

ping 8.8.8.8 -c 3
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:~#

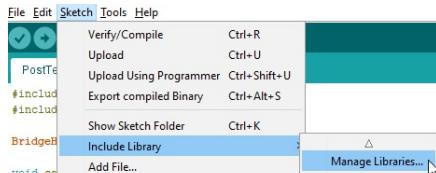
```

Autoscroll

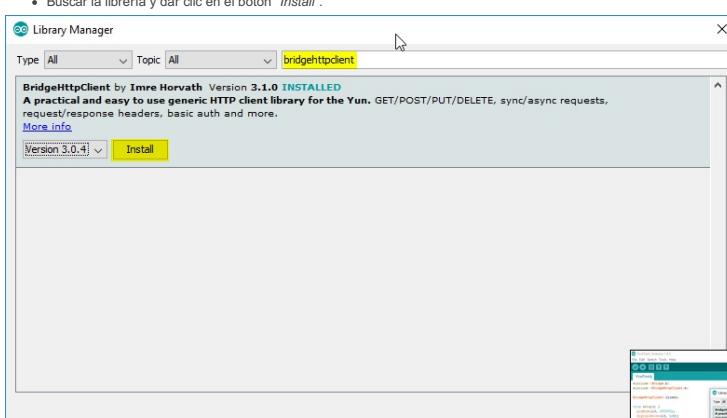
Enviar datos desde el Arduino hacia el OCB

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:

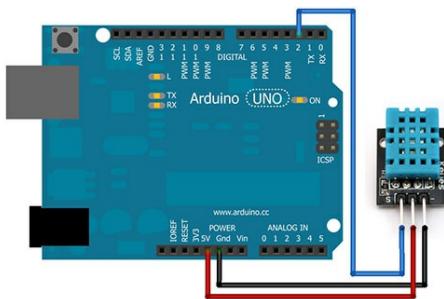
COM6 (Arduino Yún)

```

Sending request.....
Response Body:
Response Code: 201

```

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 YUN - 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 hic = 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(hic);
  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 tu turno!

Sigue la siguiente tarea para 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.

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

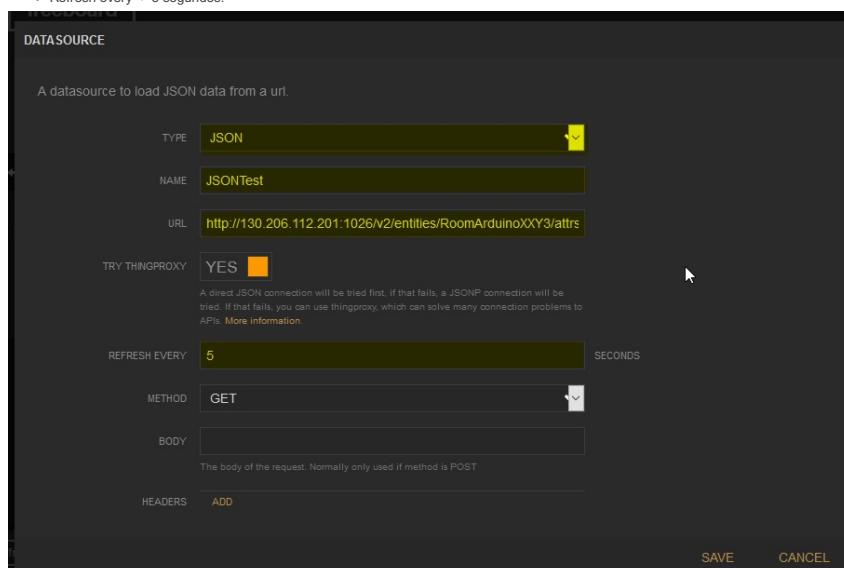
Para poder visualizar los datos capturados con la ayuda del Arduino vamos a utilizar una herramienta llamada **Freeboard**. Para hacerlo seguiremos los siguientes pasos:

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*.

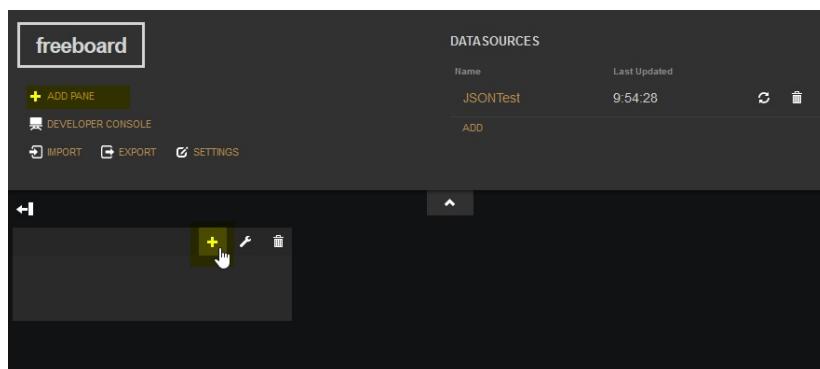
3.- En la forma se tiene que elegir *Add* en **DATASOURCES** con la siguiente información:

- 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

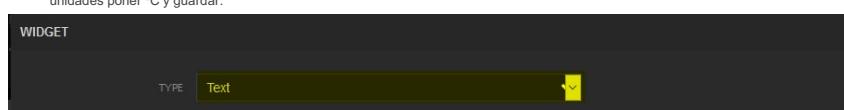


Por simplicidad hemos elegido un mecanismo de actualización **por poleo**. Nuestro panel lanzará una consulta cada 5 segundos al OCB para actualizar los valores de temperatura y humedad. Una forma alterna, frecuentemente preferida, es que el OCB le notifique al panel cada vez que recibe un nuevo valor. Para implementar esta alternativa, llamada **por suscripción**, 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 más que aparece abajo.



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





¡Ahora es tu turno!

Tu siguiente tarea es integrar ambas funcionalidades para lograr que el Arduino manda 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.

4.- Fiware Lab Cloud

En el ecosistema de Fiware, existen nodos que nos permiten desplegar los distintos componentes (llamados *Generic Enablers* en la nube para experimentar con esta arquitectura sin invertir en infraestructura.

Estos nodos conforman lo que se conoce como el **Fiware Lab** y han sido esenciales para el crecimiento de Fiware.

En esta sección se muestra cómo **desplegar una instancia del Orion Context Broker** en la nube Fiware Lab Cloud.

Para poder desplegar una instancia en Fiware Lab se deben de seguir estos pasos:

1. Entrar a <https://cloud.lab.fiware.org/auth/login/?next=/>
2. Crear una cuenta
3. Responder al correo dejando el asunto intacto
4. Esperar ... (lo sentimos!)
5. Una vez que se tiene la cuenta de Fiware Lab, iniciar sesión
6. Ir a Compute -> Instances -> Lanzar Instancia.

Nombre de la instancia	Nombre de la imagen	Dirección IP	Tamaño	Por de claves	Estado	Zona de Disponibilidad	Tarea	Estado de energía	Tiempo desde su creación	Actions
OCBTL	orion-psb-image-R5.2	192.168.226.126 IPs flotantes: 130.206.112.201	m1.small	-	Activo	nova	Ninguno	Ejecutando	21 horas, 16 minutos	<button>Crear instantánea</button>
Vallarta-ITAM	orion-psb-image-R5.4	192.168.250.53	m1.small	ITAM-Vallarta	Activo	nova	Ninguno	Ejecutando	10 meses, 1 semana	<button>Crear instantánea</button>

2015 © FIWARE. The use of FIWARE services is subject to the acceptance of the [Terms and Conditions](#), [Privacy Policy](#) and [Cookies Policy](#).



Introducir los siguientes datos en el formulario:

- * Zona de disponibilidad: Elegir la que sea.
- * Nombre de la instancia: Elegir un nombre único.
- * Sabor: small.
- * Origen de arranque de la instancia: Arrancar desde una imagen.
- * Nombre de la imagen: orion-psb-image-R5.2 (1,4 GB).



Hacer clic en Lanzar.

Ir a Acceso y Seguridad -> Crear grupo de seguridad.

<input type="checkbox"/>	Nombre	Descripción	Actions
<input type="checkbox"/>	default	Default security group	Administrar reglas
<input type="checkbox"/>	prueba	holo	Administrar reglas

Llenar nombre y descripción.

Crear grupo de seguridad.

Administrar reglas del grupo de seguridad creado.

	Nombre	Descripción	Actions
<input type="checkbox"/>	GrupoTalentLand	Grupo de seguridad elaborado para el taller de Talent Land	Administrar reglas
<input type="checkbox"/>	default	Default security group	Administrar reglas
<input type="checkbox"/>	pruebo	hola	Administrar reglas

Displaying 3 items of False

2015 © FIWARE. The use of FIWARE Lab services is subject to the acceptance of the [Terms and Conditions](#), [Privacy Policy](#) and [Cookies Policy](#).

Agregar regla y llenar el formulario con los siguientes datos:

Administrador Reglas de Grupo de Seguridad: GrupoTalentLand (661d715a-0540-4a54-a83b-89545726ed9e)							
+ Agregar regla x Eliminar Regla							
	Dirección	Tipo Ethernet	Protocolo IP	Rango de puertos	Prefijo de IP Remota	Grupo de Seguridad Remoto	Actions
<input type="checkbox"/>	Saliente	IPv6	Cualquier	Cualquier	::/0	-	Eliminar Regla
<input type="checkbox"/>	Saliente	IPv4	Cualquier	Cualquier	0.0.0.0/0	-	Eliminar Regla

Displaying 2 items of False

2015 © FIWARE. The use of FIWARE Lab services is subject to the acceptance of the [Terms and Conditions](#), [Privacy Policy](#) and [Cookies Policy](#).

- * Regla: Regla TCP a medida.
- * Dirección: Entrante
- * Puerto abierto: Puerto
- * Puerto: 1026
- * Remoto: CIDR
- * CIDR: 0.0.0.0/0

Hacer clic en Añadir

Agregar regla

Regla *: Regla TCP a medida

Dirección: Entrante

Puerto abierto *: Puerto

Puerto: 1026

Remoto *: CIDR

CIDR: 0.0.0.0/0

Descripción:

Los reglas definen el tráfico permitido a las instancias asociadas al grupo de seguridad. Una regla de un grupo de seguridad contiene tres partes principales:

Regla: Puede especificar una plantilla de reglas deseada o usar reglas TCP, UDP e ICMP personalizados.

Puerto abierto/Rango de puertos: Para las reglas de TCP y UDP puede optar por abrir un solo puerto o un rango de ellos. La opción "Rango de puertos" le proporcionará el espacio para especificar tanto el puerto de comienzo como de final del rango. Para las reglas de ICMP por el contrario debe especificar el tipo y código ICMP en los espacios proporcionados.

Remoto: Debe especificar el origen del tráfico a permitir a través de esta regla. Lo puede hacer bien con el formato de un bloque de direcciones IP (CIDR) o especificando un grupo de origen (Grupo de Seguridad). Al seleccionar un grupo de seguridad como origen se permitirá que cualquier instancia de ese grupo de seguridad pueda acceder a cualquier otra instancia a través de esta regla.

Ir a Acceso y Seguridad -> IPs flotantes -> Asignar IP al proyecto.

<https://i.imgur.com/0J4QfLa.PNG>

Cloud -> Compute -> Instancias -> Crear

De la IP asignada, clic en asociar

Cloud Store Mashup Data Account Help&Info

marinisoac1@gmail.com marinisoac1.cloud

Acceso y seguridad

Grupos de seguridad Pares de claves IPs flotantes Acceso a la API

Rsignar IP al proyecto (Cuota superada) Liberar IPs Flotantes

	Dirección IP	Dirección de IP fija asignada	Pool	Estado	Actions
<input type="checkbox"/>	130.206.112.201	-	public-ext-net-01	Activo	Asociar

Displaying 1 item of False

2015 © FIWARE. The use of FIWARE Lab services is subject to the acceptance of the [Terms and Conditions](#), [Privacy Policy](#) and [Cookies Policy](#).

Seleccionar la instancia creada previamente y dar clic en asociar

Cloud Store Mashup Data Account Help&Info

marinisoac1@gmail.com marinisoac1.cloud

Gestionar asociaciones de IP flotantes

Dirección IP *

Dirección IP *

130.206.112.201

Puerto a asociar *

OCBTL-192.168.226.126

Selecione la dirección IP que quiere asociar con una determinado instancia o puerto.

Rsignar IP al proyecto (Cuota superada) Liberar IPs Flotantes

	Estado	Actions
	Activo	Asociar

2015 © FIWARE. The use of FIWARE Lab services is subject to the acceptance of the [Terms and Conditions](#), [Privacy Policy](#) and [Cookies Policy](#).

Después de haber hecho estos pasos estará levantado el OCB en Fiware Lab Cloud para poder ser utilizado públicamente.

Felicidades! Ahora se ha familiarizado con las principales funcionalidades del Componente Orion Context Broker.