

Taller FIWARE para Ciudades Inteligentes

Prototipo de un Sistema de Alerta Sísmica

Mexicanas del Futuro - Caravanas IPN

Agosto, 2018

1. Introducción

La Internet de las Cosas (IoT, *Internet of Things*) tendrá un profundo impacto en prácticamente todas las actividades de nuestra vida.

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.

Como aplicación *inteligente* implementaremos una maqueta de un Sistema de Alerta Sísmica (SAS) que estará conformado por:

- Sensores de movimiento conectados a placas Arduino para simular movimientos telúricos
- Un nodo colector de datos (el Orion Context Broker de Fiware)
- La lógica para discriminar si se debe reportar un sismo, o simplemente registrar el movimiento telúrico
- Una estación de monitoreo que reportará con un código de colores la presencia o ausencia de sismos.

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 de movimiento con una tarjeta Arduino

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.

3. Sistema de Alerta Sísmica

La siguiente figura muestra el diagrama del Sistema de Alerta Sísmica de México.

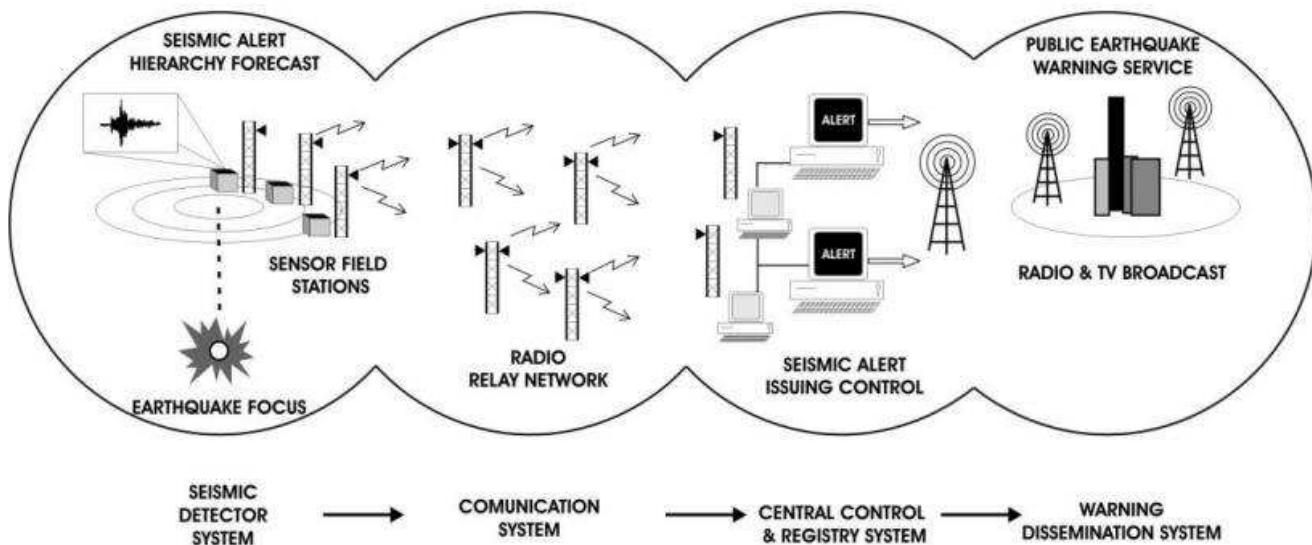


Figure 1.1 Seismic Alert System of Mexico diagram

Cuando inicia un temblor, se generan dos tipos de onda. Las ondas P se propagan muy rápidamente pero casi nunca generan daños. Las ondas S viajan mucho más despacio y son las que provocan los desastres si tienen la energía suficiente.

En un movimiento telúrico, los *{\it Field Sensors}* (FS) detectan las ondas P (de hecho, la diferencia entre las ondas P y S), hacen un primer procesamiento y mandan información a través de la red de comunicaciones, al sistema de control central.

- *En este taller, los FS se simularán con unos sensores de movimiento inercial (IMU) minIMU-04 conectados a una placa Arduino YÙN donde se hará el procesamiento inicial y, si se detecta movimiento, se enviará al sistema de control a través de una conexión WiFi.*

El sistema central analiza la información de los sensores junto con otra información y decide si debe emitir la alerta sísmica. Por ejemplo, en el SAS de México, la alerta se emite sólo si al menos cinco FS en la misma zona detectaron el movimiento y si éste es mayor de 4.5 grado Richter.

- *En este taller, el sistema central estará simulado por el Orion Context Broker, que recibe la información de los sensores a través de Arduino, y de la “lógica de negocio” implementada en un programa que decide si debe emitir una alerta cuando al menos tres sensores de la misma zona reportan un movimiento considerable.*

Si la alerta debe activarse, el sistema de control central envía notificaciones a las estaciones de radio y TV de la Ciudad de México, a proveedores de telefonía celular para aquellos abonados que tengan la aplicación, y a otros centros de difusión.

- *En este taller, la alerta se desplegará en una estación de monitoreo con un código de colores por zona:*
 - *Verde si en la zona ningún FS detectó movimientos*
 - *Amarillo si uno o dos FS en la zona detectaron movimientos*
 - *Rojo si al menos tres FS en la zona detectaron movimientos*

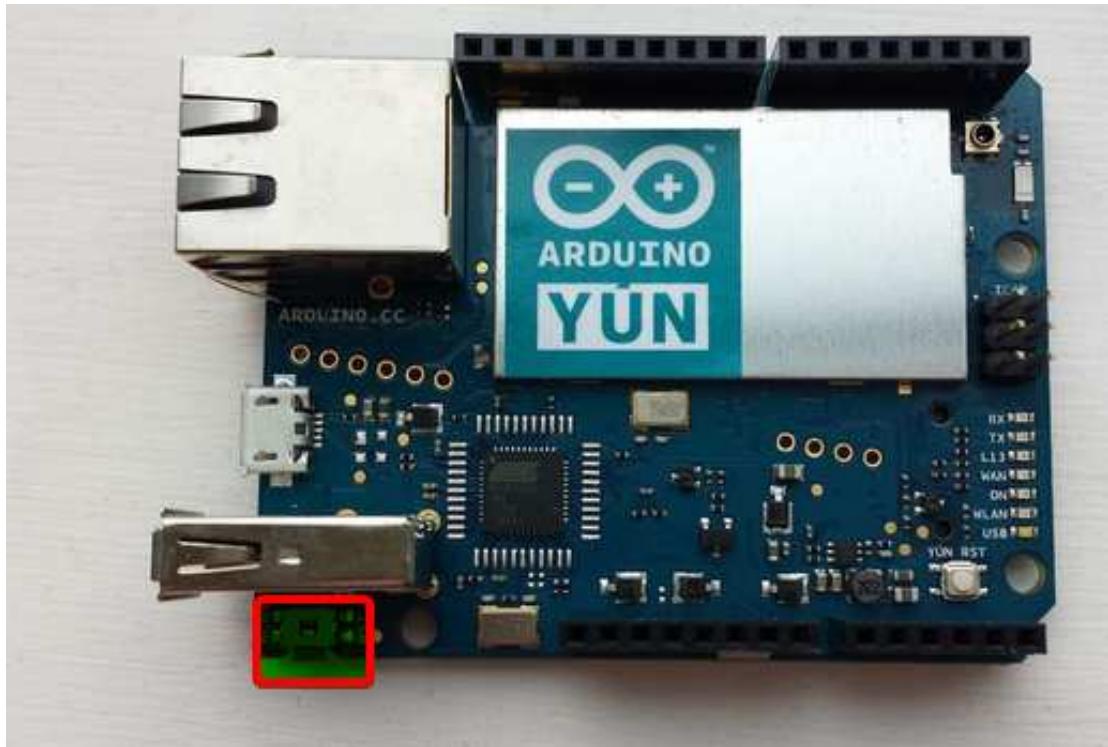
4. Desarrollo

4.1 Placa Arduino y sensor.

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 siguiente muestra la tarjeta Arduino YÙN, 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.

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. Como hemos mencionado, en este taller usaremos el IMU04 para simular un *Field Sensor* del Sistema de Alerta Sísmica.

4.1.1. 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.

- (1) Conecta la placa a un puerto USB de tu computadora. Se deberá encender un LED verde indicando que la placa está energizada (aproximadamente un min).
- (2) Ahora vamos a conectar el sensor de movimiento a la placa Arduino como se muestra en la figura siguiente:

FALTA UNA FOTO CON LA PLACA



```
boolean comma = false;  
  
void loop() {  
    // copy from serial  
    int c = SERIAL.read();  
    if (c != -1) {  
        if (comma)  
            if (c == ',')  
                comma = false;  
        } else  
            SERIAL.write(c);  
    }  
}
```

Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 Firmware Updater
Board: "Arduino Yún"
Port: "COM6 (Arduino Yún)"
Get Board Info
Programmer: "AVRISP mkII"
Burn Bootloader

rom USB-CDC
Serial ports
COM6 (Arduino Yún)
enter in comm
erwise write

Al final de este tutorial, en la sección *Sketch Sensor IMU encontrarás un programa que contiene una serie de funciones para que no invirtamos demasiado tiempo con los detalles de la programación y configuración de los sketch.

- (4) Selecciona File -> New. Aparece una ventana para escribir un nuevo sketch

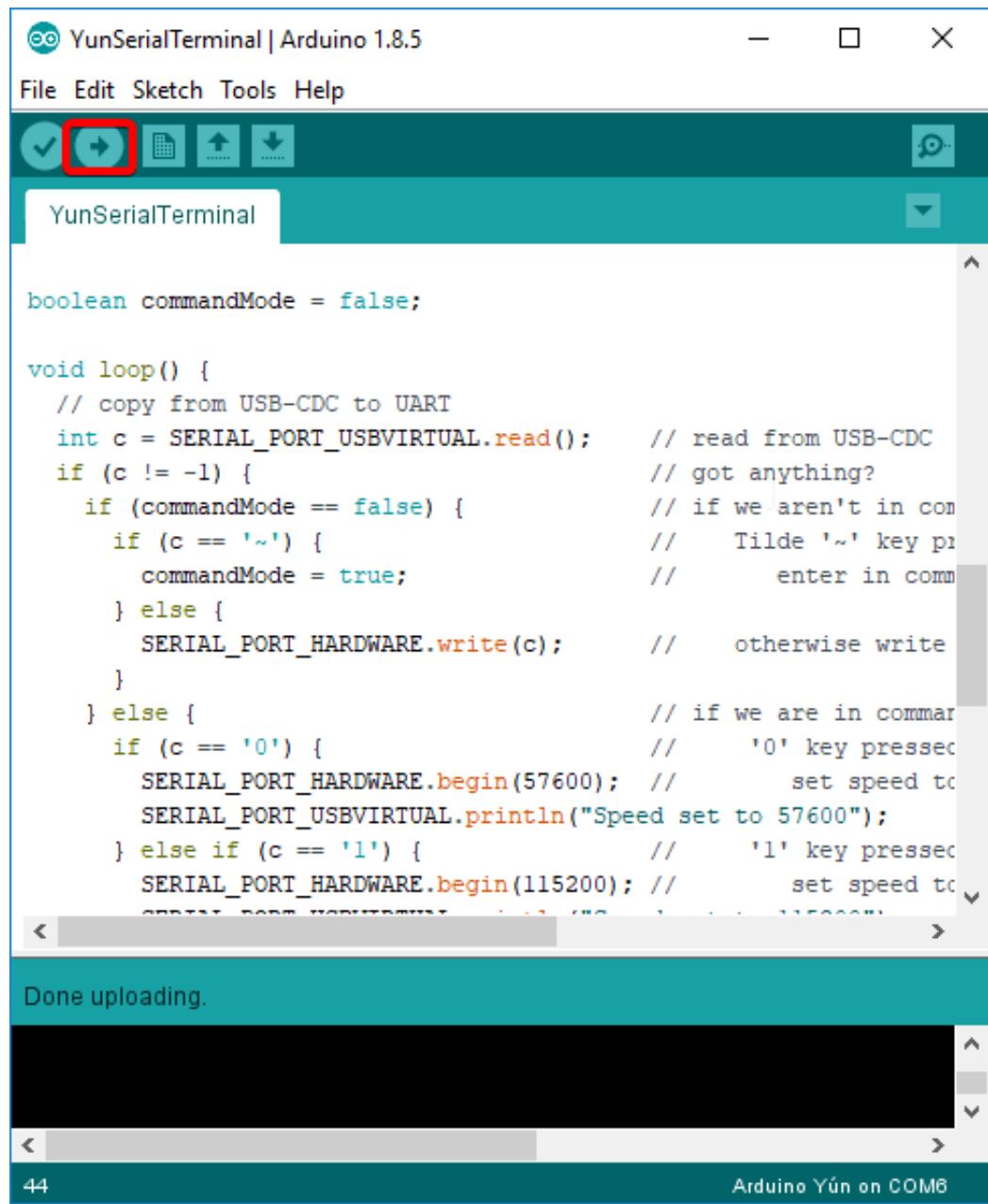
```
sketch_aug24b Arduino 1.8.5
Archivo Editar Programa Herramientas Ayuda
sketch_aug24b
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
Arduino Yun en COM3
```

- (5) Borra su contenido y sustitúyelo por el código del Sketch Sensor IMU.
- (6) Ahora sigue las indicaciones del instructor para que puedas configurar tu placa con los parámetros de tu zona (identificador de FS y zona) y para que puedas hacer un pequeño programa que lee los movimientos del sensor.
- (7) Da clic en la flecha que se encuentra arriba al lado izquierdo para compilar y transferir este sketch a la tarjeta. Espera a que se vea el mensaje *Done uploading*.



The screenshot shows the Arduino IDE interface with the title bar "YunSerialTerminal | Arduino 1.8.5". The menu bar includes File, Edit, Sketch, Tools, and Help. The toolbar has several icons, with the upload icon (a right-pointing arrow) highlighted by a red box. The main code editor window contains the "YunSerialTerminal" sketch. The code is as follows:

```
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 pressed
                commandMode = true;                // enter in command mode
            } else {
                SERIAL_PORT_HARDWARE.write(c);     // otherwise write it
            }
        } 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");
            }
        }
    }
}
```

The status bar at the bottom left says "Done uploading." and the bottom right says "Arduino Yún on COM6".

- (8) Ahora da clic en la lupa que se encuentra arriba del lado derecho para abrir el monitor serie.

The screenshot shows the Arduino IDE interface with the following details:

- Title Bar:** YunSerialTerminal | Arduino 1.8.5
- Toolbar:** File, Edit, Sketch, Tools, Help, and a redboxed Refresh icon.
- Sketch Area:** The code for `YunSerialTerminal` is displayed. It includes logic for reading from the USB-CDC port and writing to the SERIAL_PORT_HARDWARE.
- Status Bar:** Done uploading.
- Bottom Status:** Sketch uses 5072 bytes (17%) of program storage space. Maximum is 28600. Global variables use 453 bytes (17%) of dynamic memory, leaving 2107.
- Page Number:** 44
- Page Footer:** Arduino Yún on COM6

- (9) Toma la tarjeta protoboard donde se encuentra el sensor y agítala levemente, o un poco mas fuerte. Podras observar en el monitor que cada segundo se toma una lectura del sensor y ésta se reporta al monitor en caso de que haya detectado movimientos.

¡FELICIDADES. Ya tienes un nodo sensor de movimiento funcionando!

4.1.2. Conexión de la placa Arduino YÙN a la red WiFi/Internet del laboratorio

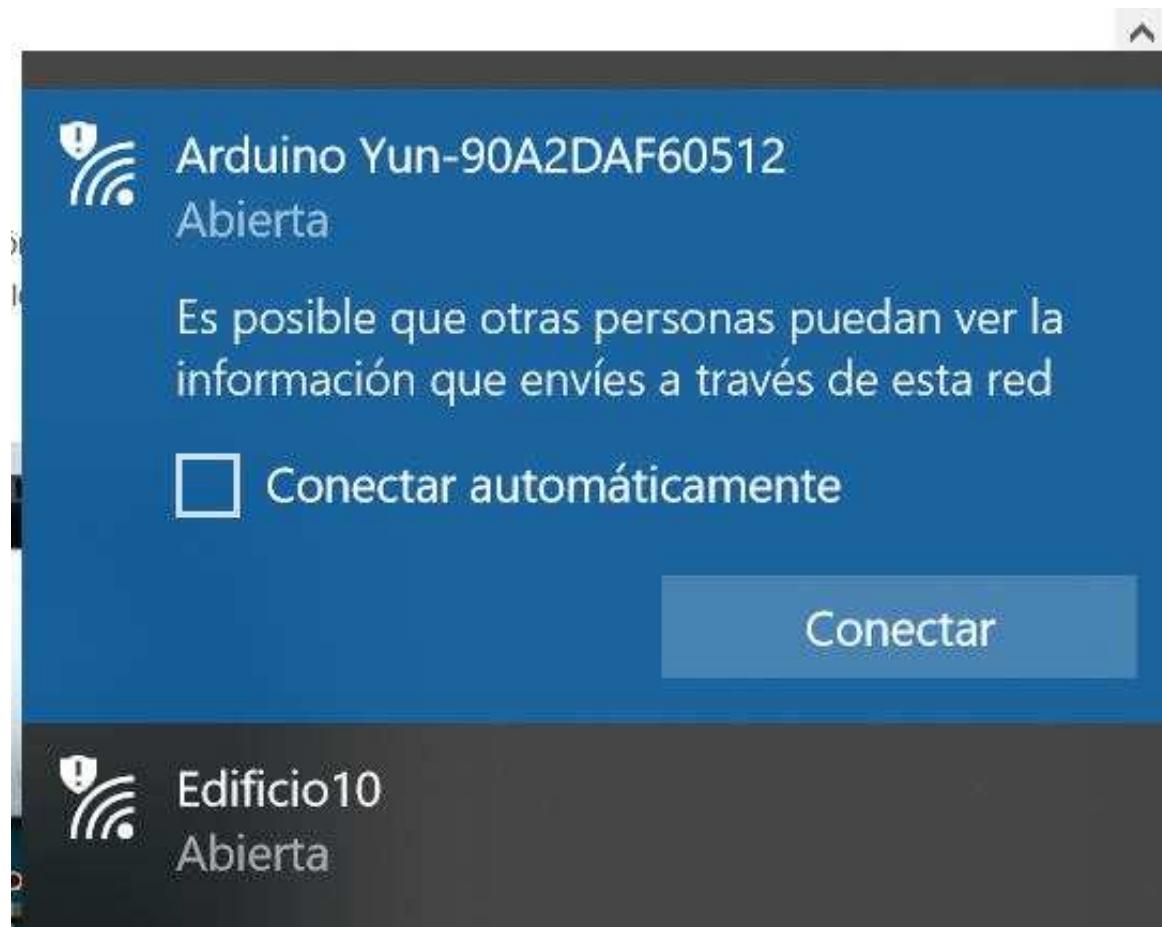
Para poder enviar las lecturas del sensor a nuestro sistema de control, primero debemos conectar

la placa Arduino a la red local WiFi del laboratorio. A través de esta red podremos enviar las lecturas al sistema de control.

Nuestra placa YÙN contiene un puerto WiFi al que se pueden conectar otras computadoras. Nos vamos a conectar a través de él para indicarle que en realidad queremos que la placa se conecte a la red del laboratorio.

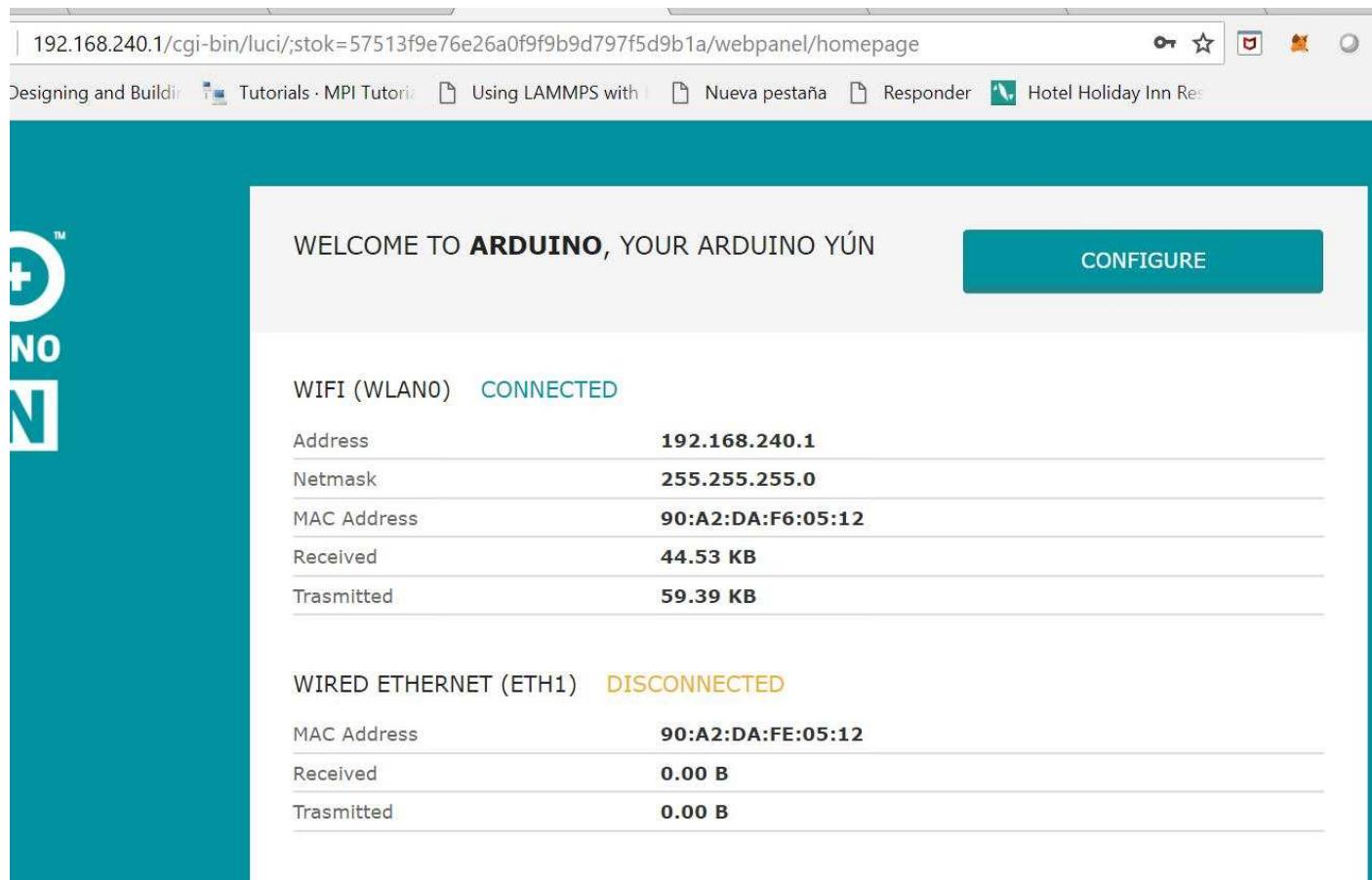
El siguiente paso puede ser un poco confuso porque no conocemos qué red pertenece a qué placa. Podemos apagar todas las placas e ir configurando una a una, o simplemente podemos ir acoedando qué equipo va a configurar qué placa.

- (1) En tu computadora portátil (o desde tu celular), encuentra un identificador de red WiFi de un Arduino. Se identifica por *Arduino Yun-XXXXXXXXXXXX* (las XX representan números hexadecimales).
- (2) Haz que tu dispositivo se conecte a esa red.



- (3) Ahora que tu dispositivo está en la red del Arduino, conéctate a un pequeño servidor que tiene la placa para configurarlo. En un navegador, en la pestaña del URL escribe 192.168.240.1
- (4) Aparecerá una pantalla solicitando una contraseña. Escribe **arduino** y en nueva

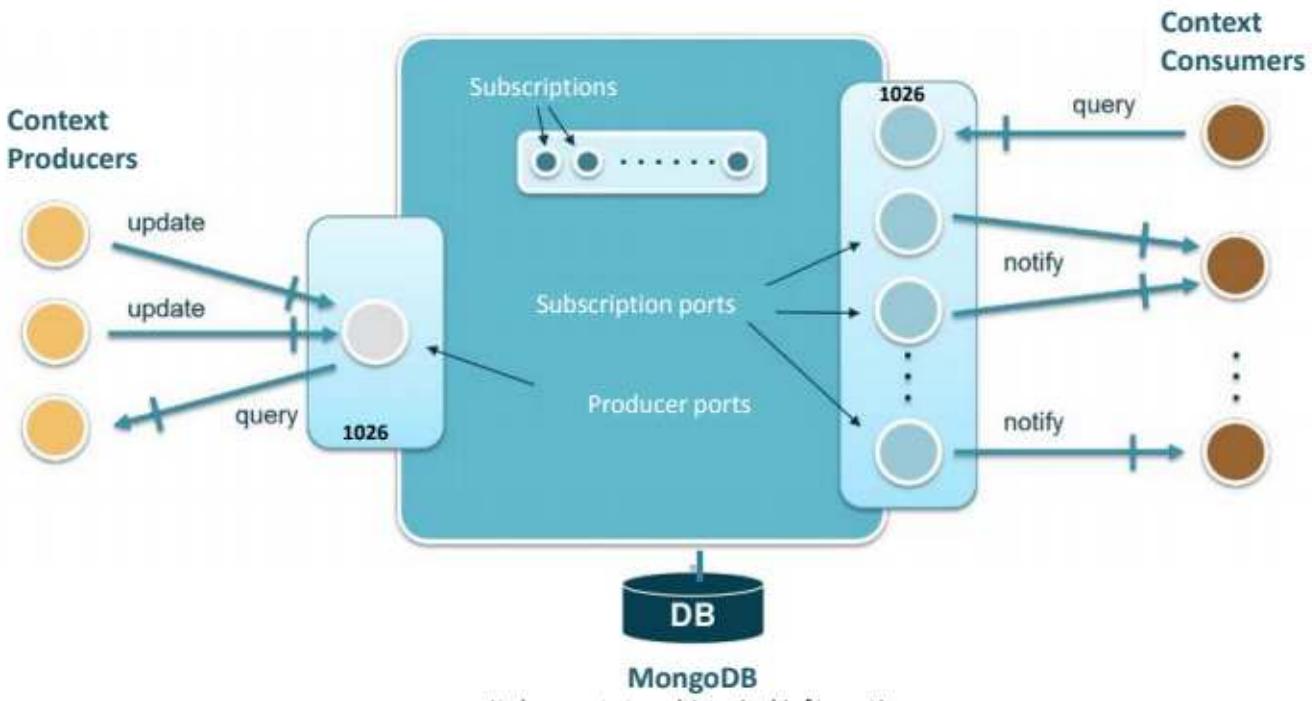
pantalla, selecciona el botón **CONFIGURE**.



- (5) En la ventana de configuración, en la parte inferior *WIRELESS PARAMETERS* selecciona el menú de *DETECTED WIRELESS NETWORKS* y da clic en la red del Laboratorio (el instructor te indicara cuál es).
- (6) Ahora da clic en *CONFIGURE & RESTART* y espera a que la nueva configuración se haya completado.
- (7) Mientras tanto, conecta también tu dispositivo a la red del Laboratorio.
- (8) Sigue las indicaciones de tu instructor para modificar tu sketch para que envíe las lecturas al Orion Context Broker (OCB), quien funge como el punto de entrada del sistema de control.

4.2. Envío de lecturas al Orion Context Broker

En el **Apéndice** de este tutorial encontrarás un poco de información sobre el *Orion Context Broker* (OCB) de Fiware. Por ahora es suficiente con señalar que el OCB es un componente esencial de la plataforma. Es un intermediario entre los productores de datos (nuestro sensor y la placa Arduino) y los consumidores de éstos (nuestro programa para implementar la lógica que decide si hay que activar o no una alerta, y el tablero de mando).



La información se envía y se consulta del OCB en formato JSON siguiendo una especificación muy precisa, a través de una interfaz de programación (API, *Application Programming Interface*).

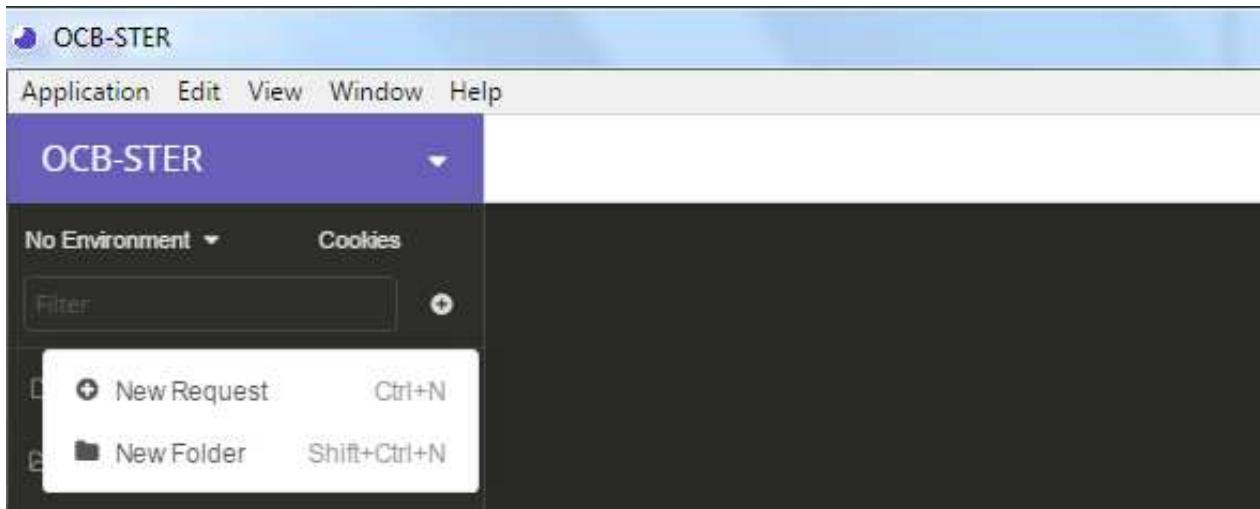
Cuando agitamos el sensor de movimiento, nuestro Arduino no sólo reporta la magnitud en el monitor serie; también envía un mensaje al OCB en donde se almacena el último dato enviado por cada estación de monitoreo.

En este taller solamente consultaremos al OCB para verificar que las lecturas (la última lectura) que envía nuestro sensor, efectivamente se registran en él. Para ello, vamos a utilizar llamado *Insomnia* que nos permite enviar comandos del protocolo HTTP (*Hypertext Transfer Protocol*), que es como se interactúa con el OCB.

Para poder interactuar con el OCB utilizaremos la herramienta [Insomnia](#). Si lo deseas, puedes utilizar cualquier otro cliente REST.

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

- (1) Abre el programa *Insomnia* y da clic en **New Request**



- (2) Asigna un nombre a tu consulta (por ejemplo, consulta del equipo xx). Observa que por default, el tipo de comando que te propone Insomnia es GET.
 - (3) 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` , donde las XX.XX se sustituyen por la dirección IP que te indique el instructor.

Observa del lado derecho el resultado de la consulta.

The screenshot shows the OCB-STER application interface. The top navigation bar includes 'Application', 'Edit', 'View', 'Window', and 'Help'. Below the navigation is a toolbar with 'OCB-STER' (selected), 'No Environment', 'Cookies', and a 'Body' dropdown set to 'GET' with the URL 'http://127.0.0.1:1026/v2/entities'. To the right of the URL are 'Send', '200 OK', 'TIME 140 ms', and 'SIZE 655 B'. The main area has tabs for 'Auth', 'Query', 'Header', and 'Docs'. On the left, a sidebar lists 'Operaciones comunes' with 'POST inserta-entidad' and 'GET obtiene-todas-entidades'. The right side shows a 'Preview' tab displaying JSON data for entities, including fields like 'frequency', 'voltage', and 'metadata'.

Consulta acotada.

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

- (4) Haz una nueva consulta o modifica la que ya tienes. Lo único que debes hacer es extender el URL con el ID de tu entidad

Aquí necesito el JSON para hacer una consulta acotada específica a la zona

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

- (5) Agita tu sensor de movimiento para asegurar que llegó una notificación al OCB y lanza nuevamente la consulta.

Rpite varias veces la consulta agitando con distinta intensidad tu sensor.

4.3 El sistema de control

Por falta de tiempo, tus instructores han implementado para tí la “lógica de negocio” del SAS que describimos en la introducción. Básicamente, se trata de un programa que toma las lecturas del OCB y verifica si algún sensor reporta un movimiento telúrico. De ser así, y si el movimiento rebasa un cierto umbral, lo reporta en la estación de monitoreo representando el fenómeno con un color.

Si es uno o dos sensores de la misma zona, ésta cambiará de color Verde a color Amarillo, pero si son tres o más los sensores que detectaron un movimiento considerable, entonces la zona se colorea de Rojo y simboliza que se está emitiendo una alerta sísmica.

- Identifica quiénes son tus compañeras de zona y coordínate con ellas para que consigan hacer pasar su zona de Verde a Amarillo y finalmente a Rojo.

Conclusión

Hemos llegado al final de nuestro taller. Esperamos que hayas disfrutado tu estancia con nosotros y, aunque tengas todavía algunas dudas por lo breve que fue este taller, esperamos que hayas podido asimilar los principios básicos de operación de las plataformas para Internet de las Cosas y de cómo las Tecnologías de Información y de Comunicaciones nos ayudan a salvar miles de vidas en casos de desastres naturales como son los terremotos.

¡MUCHAS FELICIDADES! Lograste mucho en muy poco tiempo. ¿Te imaginas todo lo que puedes hacer al explorar un poquito más estas tecnologías? ¡No dudes en contactarnos si tienes alguna duda!

Apéndice.- Orion Context Broker

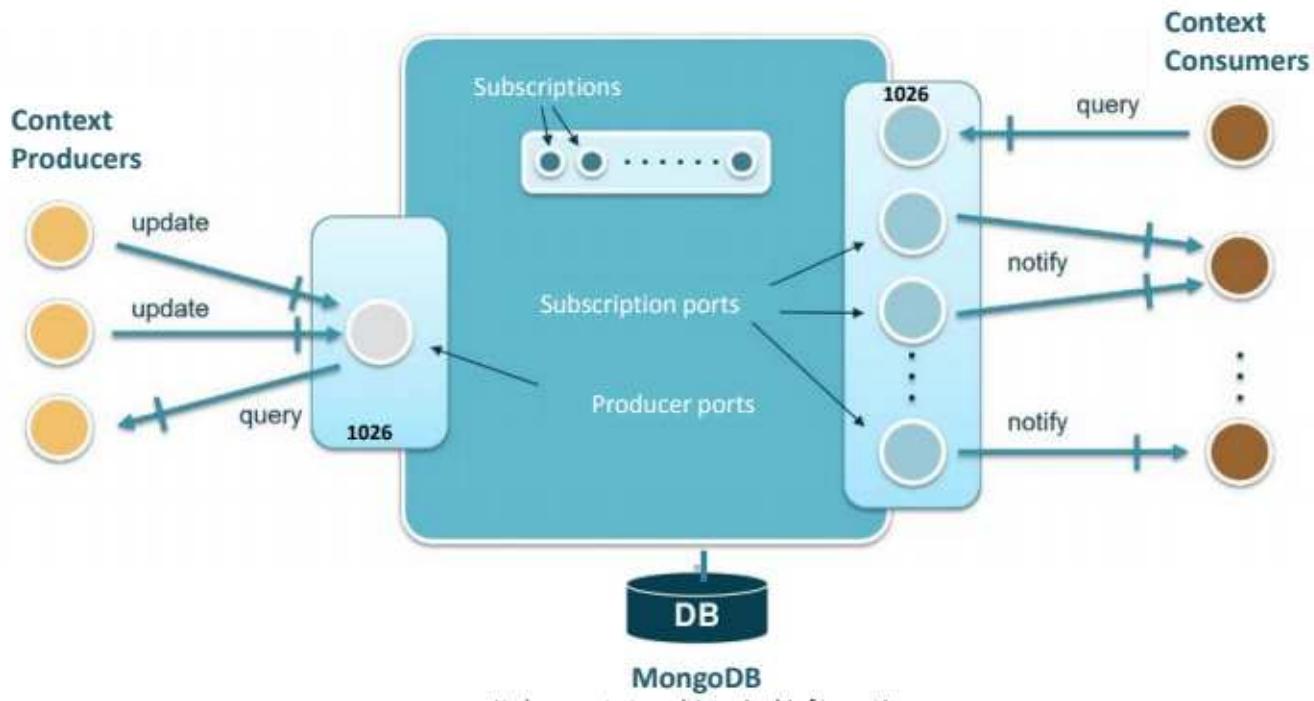
En Fiware, para que las aplicaciones puedan obtener información de los sensores y objetos

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 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; 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.

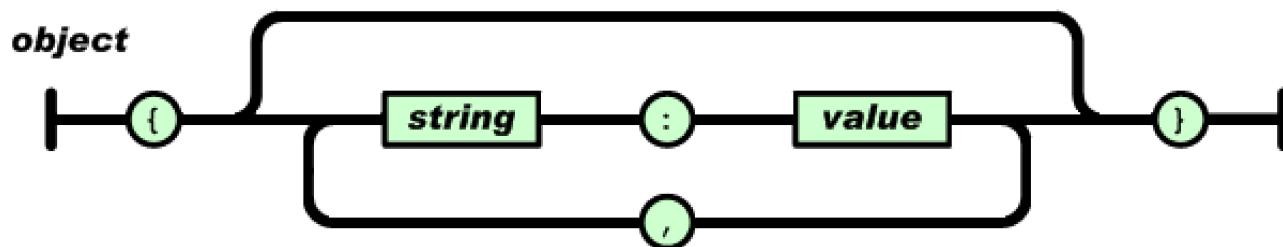
Representación de datos de contexto

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 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 las 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**.

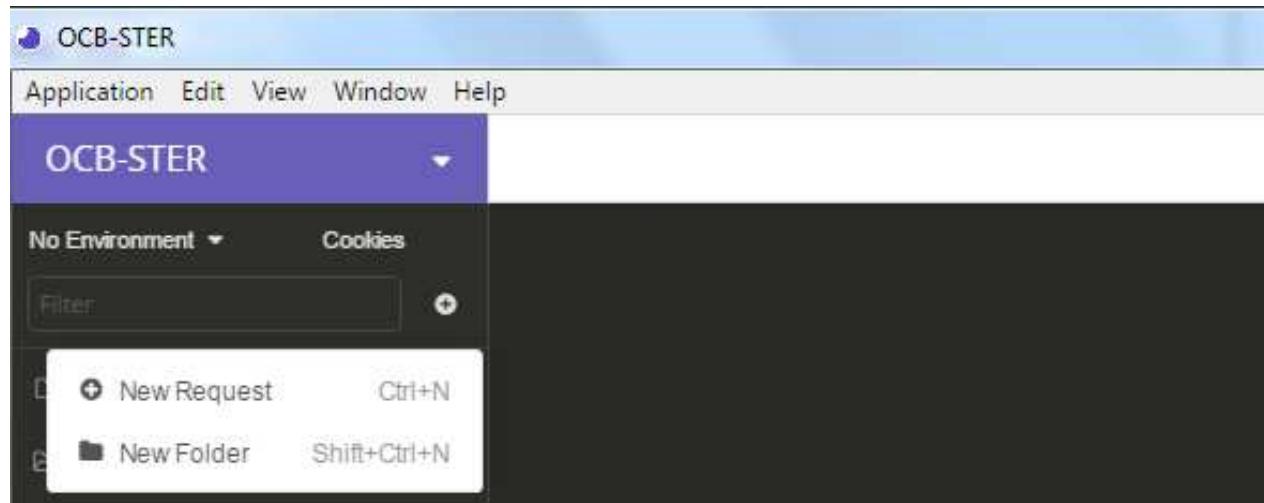
Para indicar que se está enviando información se debe de poner **Content-Type** y para indicar que se desea recibir se debe de poner **Accept**.

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.

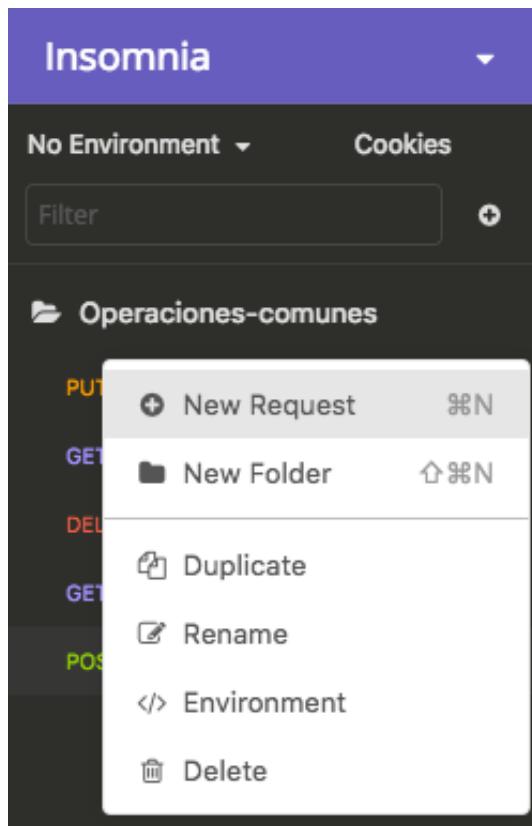
Crearemos en insomnia una carpeta llamada **Operaciones-Comunes**. En esta carpeta se guardarán todas las consultas que hagamos.



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.

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.

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 OCB-STER interface with the following details:

- Title Bar:** OCB-STER – obten-todas-entidades
- Menu Bar:** Application Edit View Window Help
- Toolbar:** OCB-STER, GET http://127.0.0.1:1026/v2/entities, Send, 200 OK, TIME 140 ms, SIZE 655 B
- Left Sidebar:** No Environment, Cookies, Filter, Operaciones comunes, POST inserta-entidad, GET obten-todas-entidades (selected).
- Central Area:** Body, Auth, Query, Header, Docs.
- Right Area:** Preview, Header, Cookie, Time. The preview shows JSON data representing multiple entities, including frequency and voltage attributes.

Consulta acotada.

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

The screenshot shows the OCB-STER interface with the following details:

- Title Bar:** OCB-STER – obten-una-entidad
- Menu Bar:** Application Edit View Window Help
- Toolbar:** OCB-STER, GET http://127.0.0.1:1026/v2/entities/Med025, Send, 200 OK, TIME 16 ms, SIZE 16 B
- Left Sidebar:** No Environment, Cookies, Filter, Operaciones comunes, POST inserta-entidad, GET obten-todas-entidades, GET obten-una-entidad (selected).
- Central Area:** Body, Auth, Query, Header, Docs.
- Right Area:** Preview, Header, Cookie, Time. The preview shows JSON data for a single entity named Med025, including its type (LineMeter) and current value (2.305).

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/`.

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
}
```

PUT http://130.206.112.201:1026/v2/entities/Room01/attrs/temperature Send
204 No Content TIME 188 ms SIZE 0 B

```

1 + {
2   "value":100,
3   "type":"Float"
4 }

```

No body returned for response

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

GET http://130.206.112.201:1026/v2/entities Send
200 OK TIME 188 ms SIZE 139 B

```

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
  }
}
```

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

Sketch Sensor IMU

```
* Lee datos del IMU04 y los reporta
* como si fueran generados por un Field Sensor en
```

```

* un Sistema de Alerta Sismica
*/



#include <Time.h>
#include <TimeLib.h>
#include <Wire.h>
#include <LSM303.h>

LSM303 compass;
char report[80];
int ejex, ejey, ejez, respuesta;
int mascara = 65520; // en binario 111111111110000
HttpClient client;
String var, add, method, header, json;
bool bandera;

void setup()
{
    Serial.begin(9600);
    Bridge.begin();
    Wire.begin();
    compass.init();
    compass.enableDefault();
    registrarSensor();
}

void loop()
{
    compass.read();

    ejex = (compass.a.x & mascara)>>4;
    ejey = (compass.a.y & mascara)>>4;
    ejez = (compass.a.z & mascara)>>4;

    if(ejex<0)
        ejex = -ejex;
    if(ejey<0)
        ejey = -ejey;
    if(ejez<0)

```

```
    ejez = -ejez;
Serial.println(ejex);
Serial.println(ejey);

comprobacion();
delay(1000);
}

void comprobacion(){
if(ejex >= 700){
    Serial.println("temblor trepidatorio sobre x");
    enviarDatosTremblor(ejex,"Ocilitorio");
    bandera = false;
}

if(ejey>= 700&& bandera){
    Serial.println("temblor trepidatorio sobre y");
    enviarDatosTremblor(ejey,"Trepidatorio");

}
bandera = true;
}

void enviarDatosTremblor(int magnitud, String tipo){
Process p;
/*String dia, mes, year, hora, minutos;
dia = day();
mes = month();
year = year();
hora = hour();
minutos = minutes();*/
p.begin("curl");
p.addParameter("-H");
p.addParameter("Content-Type: application/json");
p.addParameter("-X");
p.addParameter("PUT");
p.addParameter("-d");
```

```

        json = "{\"value\":\""+(String)magnitud+"\", \"metadata\":{\"tipo\": {\"value\":" + tipo + "\"}, \"fecha\":{\"value\":\"27-08-2018\"}, \"hora\":{\"value\":\"10:30\"}}}";
    Serial.println(json);
    p.addParameter(json);
    p.addParameter("-k");
    p.addParameter("http://192.168.1.72:1026/v2/entities/SensorSismico2/attrs/temblor");
    p.run();
    while (p.available() > 0) {
        char c = p.read();
        Serial.print(c);
    }
    Serial.flush();
    p.close();
}

void registrarSensor(){
    Process p;
    Serial.println("Registro de sensor");
    p.begin("curl");
    p.addParameter("-H");
    p.addParameter("Content-Type: application/json");
    p.addParameter("-X");
    p.addParameter("POST");
    p.addParameter("-d");
    p.addParameter("{\"id\": \"SensorSismico2\", \"type\": \"SensorSismico\", \"temblor\":{\"value\":\"0\", \"metadata\":{\"tipo\": {\"value\":\"-\"}, \"fecha\":{\"value\":\"-\"}, \"hora\":{\"value\":\"-\"}}}}");
    p.addParameter("-k");
    p.addParameter("http://192.168.1.72:1026/v2/entities");
    p.run();
    while (p.available() > 0) {
        char c = p.read();
        Serial.print(c);
    }
    Serial.flush();
    p.close();
    return;
}

```

