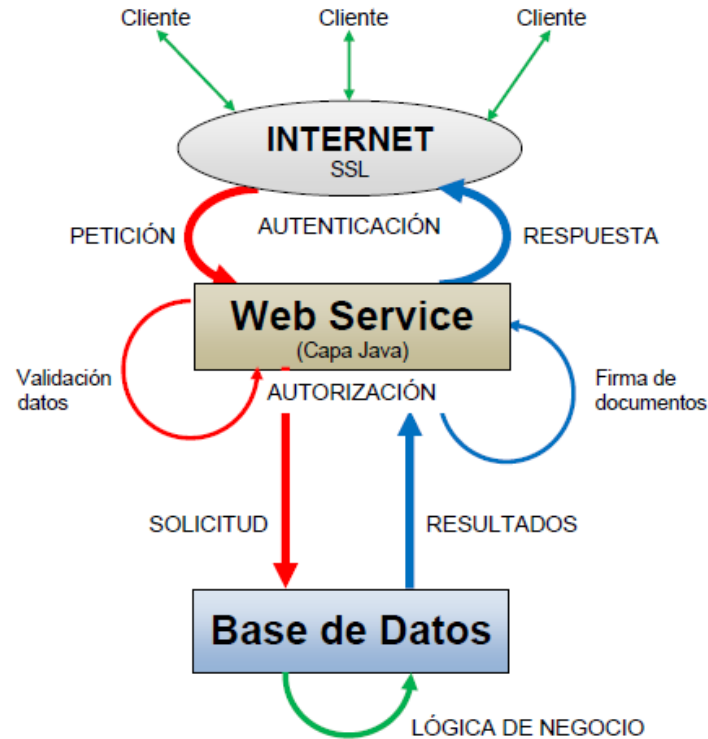


Arquitectura Corporativa de Web Services

Esquema de un modelo de Arquitectura Web Services





CUAHSI
HIS
Sharing hydrologic data

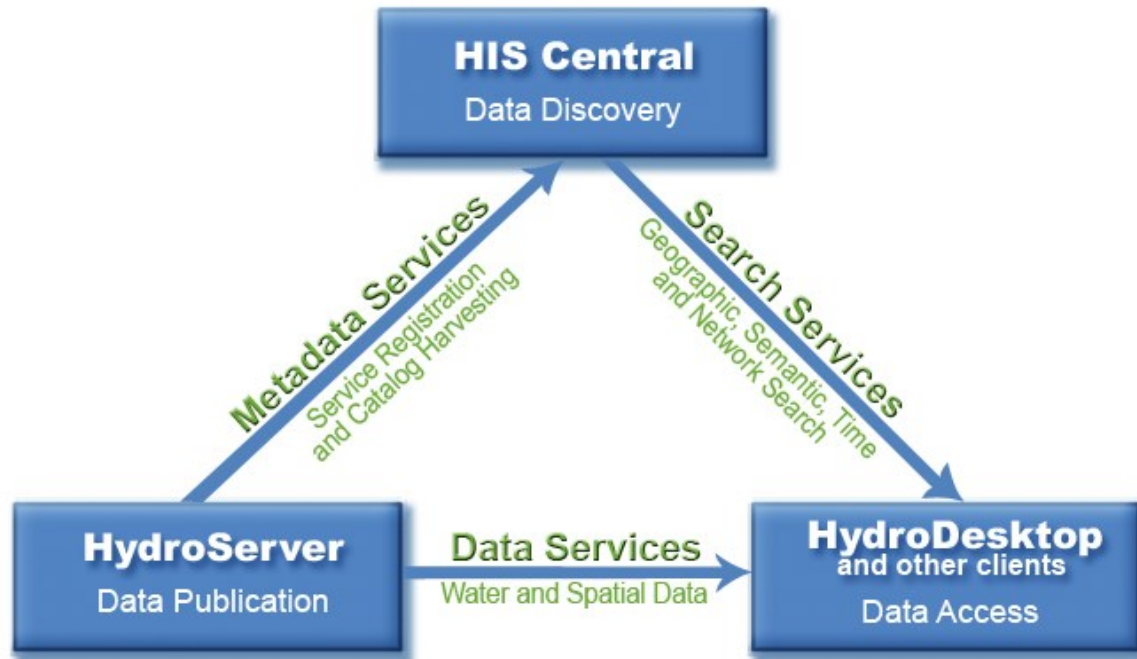
PROYECTO conducido por el investigador principal David Maidment (Universidad de Texas en Austin) y concluyó en 2011.

El Sistema de Información Hidrológica CUAHSI (HIS) es un sistema basado en Internet para el intercambio de datos hidrológicos.

Se compone de bases de datos y servidores,.

Conectados a través de servicios web a aplicaciones de cliente, lo que permite la publicación, el descubrimiento y el acceso de los datos.

Componentes clave

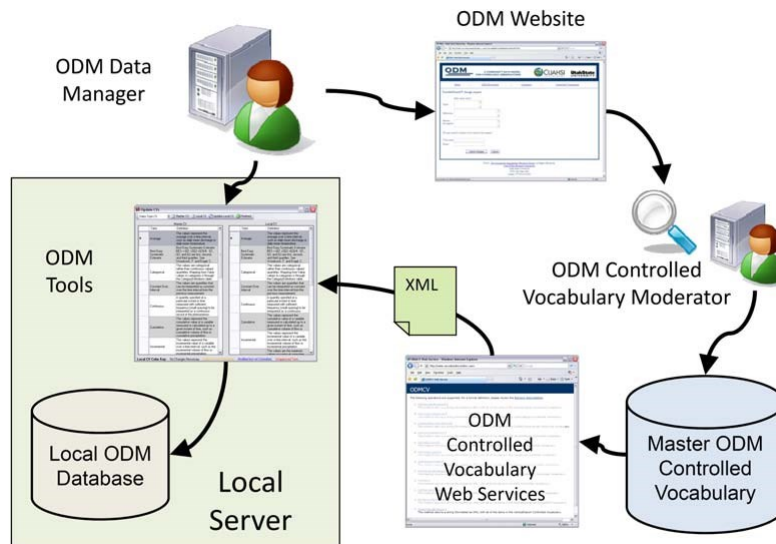


Hay tres tipos de computadoras que almacenan y procesan datos:

- 1.-**HIS central** - contiene copias de los metadatos que facilita las búsquedas; funciona como un motor de búsqueda, y permite realizar búsquedas de manera eficiente por los clientes.
- 2.-**HydroServer** - almacena, organiza y publica datos; permite coleccionar a los metadatos por el His Central y compartir con los clientes.
- 3.-**Clientes (HydroDesktop)** - ofrece a los usuarios una interfaz conveniente para acceder a los datos, recupera los metadatos de su Centro y recupera datos de HydroServers.

Así mismo tres tipos de servicios web que permiten a los ordenadores comunicarse a través de Internet:

1. **Servicios de datos** - permite que los datos del agua y los datos espaciales relacionados sea recuperado por el cliente (como HydroDesktop) y computadoras.
2. **Buscar Servicios** - permite a las computadoras cliente realizar búsquedas en el catálogo de búsqueda en His Central.
3. **Servicios de Metadatos** - permite a His Central recuperar los metadatos necesarios para construir el catálogo de búsqueda. Estos conforman el núcleo de CUAHSI-HIS.





JSON es el acrónimo para *JavaScript Object Notation*, y aunque su nombre lo diga, no es necesariamente parte de JavaScript, de hecho es un estándar basado en texto plano para el intercambio de información, por lo que se usa en muchos sistemas que requieren mostrar o enviar información para ser interpretada por otros sistemas, la ventaja de JSON al ser un formato que es independiente de cualquier lenguaje de programación, es que los servicios que comparten información por éste método, no necesitan hablar el mismo idioma, es decir, el emisor puede ser Java y el receptor PHP, cada lenguaje tiene su propia librería para codificar y decodificar cadenas de JSON.

¿Cómo se forma una cadena de JSON?

El principio básico es con pares atributo-valor, éstos deben estar encerrados entre llaves { , } que es lo que definen el inicio y el fin del objeto.

Para entender esto de mejor forma, vamos a formar un objeto JSON para los datos de una persona:

```
1 //Ejemplo de JSON para un objeto tipo Persona
2 {
3   "nombre" : "Ricardo James Aranda Valverde",
4   "edad":18,
5   "nacionalidad" : "peruano",
6   "altura":"165 cm",
7   "peso":50
8 }
```

En el ejemplo formamos un objeto JSON para una **persona** con sus datos básicos, como son su nombre, edad, nacionalidad, etc. Algo que podrán notar de entrada es que **todos los atributos y valores se encierran entre comillas**, esto es importante ya que parte del estándar de JSON establece que **un atributo debe delimitarse por comillas**, lo mismo aplica para sus valores, pero no para todos, pues como notarán, **los números pueden no tener comillas**, ya que son en sí un valor primitivo aceptado en la mayoría de los lenguajes que interpretan JSON. Lo anterior aplica tanto a números enteros, como decimales, positivos y negativos, booleanos y hasta arreglos y otros objetos.

JSON puede almacenar arreglos e incluso otros objetos, comencemos primero con un ejemplo de un arreglo, continuemos con el objeto anterior de persona, agreguémosle ahora el atributo *pasatiempos*, el cual será un listado de hobbies que tiene nuestro amigo Jean.

```
1{  
2"nombre":"Jean Quiroz",  
3"edad":18,  
4"nacionalidad":"Peruano",  
5"altura":"165 cm",  
6"peso":60,  
7"pasatiempos":["Polo","Cricket","Ski","Drafting","Gaming"]  
8}
```

El nuevo atributo *pasatiempos* tiene como valor un arreglo, para definir un arreglo, debemos enlistar sus elementos separados por coma y encerrados entre corchetes [,], de esta forma creamos un arreglo en JSON, sus valores pueden ser cualquier valor aceptado en JSON y en el lenguaje emisor/receptor (numéricos, textos, booleanos, incluso otros arreglos y otros objetos).

Finalmente, para demostrar otros dos posibles valores aceptados en el formato JSON, agregamos su dirección, el cual será un objeto con calle, número y país, y el atributo soltero, que será un booleano para indicar si es casado o no.

```
1 {  
2  "nombre": "Jean Quiroz",  
3  "edad": 18,  
4  "nacionalidad": "Peruano",  
5  "altura": "165 cm",  
6  "peso": 60,  
7  "pasatiempos": ["Polo", "Cricket", "Ski", "Drafting", "Gaming"],  
8  "soltero": true,  
9  "direccion": {  
10     "calle": "Av. Anchoqueta",  
11     "numero": "123",  
12     "pais": "Perú"  
13 }  
14 }
```

Terminología SQL y JSON

Las similitudes primarias entre SQL y JSON son las siguientes.

Base de datos SQL	Base de datos NoSql
Esquema	Namespace JSON
Tabla	Colección JSON
Registro	Documento JSON
Campo	Campo
Índice	Índice
Llave primaria	Llave primaria

Namespace JSON

Es aquel que es representado como un esquema SQL. Por ejemplo en una base de datos relacional el esquema de “Inventarios”, contendrá todas las tablas relacionadas a la información del departamento de inventarios como por ejemplo Costos, Precios, Órdenes de compras, Facturas, etc.

Colección JSON

Es un agrupador de documentos JSON. Las estructuras de los documentos dentro de una colección pueden ser diferentes. Por ejemplo puedo insertar dos documentos con diferente estructura en la colección llamada “Clientes”.

```
{ "clave"      :1001,  
  "Nombre": "Jorge Daniel"  
}
```

```
{ "clave":1001,  
  "nombre": "Jorge Daniel"  
  "dni": "01346743"  
  "ciudad": "Huaraz"  
}
```

Todos los documentos JSON tienen un `_id` único y este deber ser del mismo tipo para todos los documentos en la colección. El `_id` se genera automáticamente o puede ser especificado durante la inserción de un documento.

¿Qué es un documento JSON?

Es aquel que esta compuesto de un par de objetos, campo y valor.

Ejemplo {clave:10001, nombre:"Jorge"}. Los campos pueden estar en cualquier orden y pueden estar anidados u organizados en matrices.

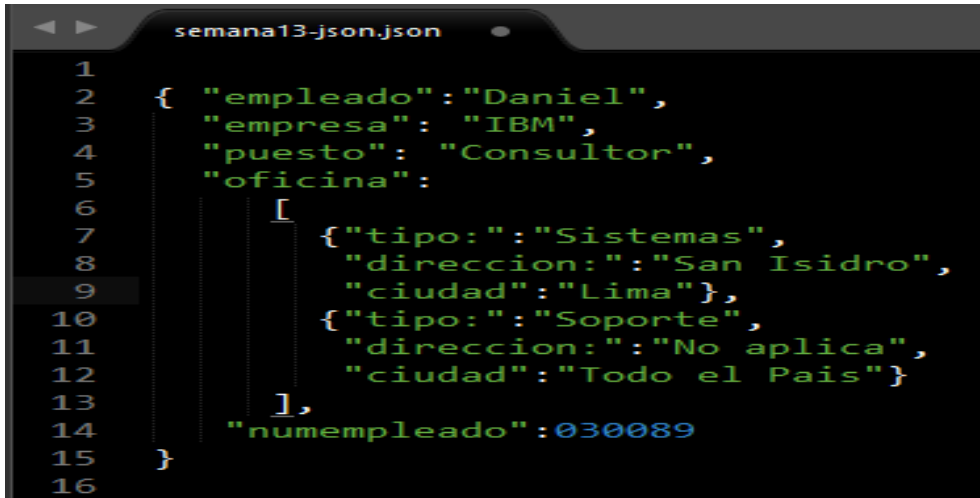
Ejemplo:

```
{  "empleado": "Daniel",
    "empresa": "IBM",
    "puesto": "Consultor",
    "oficina":
    [
      {"tipo": "Sistemas",
        "direccion": "San Isidro",
        "ciudad": "Lima"},
      {"tipo": "Soporte",
        "direccion": "No aplica",
        "ciudad": "Todo el Pais"}
    ],
    "numempleado": 030089
}
```

Los nombres de campos siempre deben ser de tipo String y deben ser únicos. Y los valores pueden ser de cualquier tipo soportados por JSON.

Se recomienda utilizar un ide de desarrollo para gestionar mejor la construcción de archivos json

CON SUBLIME



```
1 { "empleado": "Daniel",
2   "empresa": "IBM",
3   "puesto": "Consultor",
4   "oficina": [
5     { "tipo": "Sistemas",
6       "direccion": "San Isidro",
7       "ciudad": "Lima" },
8     { "tipo": "Soporte",
9       "direccion": "No aplica",
10      "ciudad": "Todo el Pais" }
11   ],
12   "numempleado": 030089
13 }
14
15
16
```

CON BRACKETS

```
Navegación  Desarrollo  Ayuda  Emmet

1  {
2    "empleado": "Daniel",
3    "empresa": "IBM",
4    "puesto": "Consultor",
5    "oficina":
6      [
7        {"tipo": "Sistemas",
8          "direccion": "San Isidro",
9          "ciudad": "Lima"},
10       {"tipo": "Soporte",
11        "direccion": "No aplica",
12        "ciudad": "Todo el Pais"}
13      ],
14    "numempleado": 030089
15  }
```

Estructura de un documento JSON

1. Los campos y sus valores siempre están en par {"articulo":"A-10340"}.
2. Los datos o campos están separados por comas {"articulo":"A-10340", "precio":2345}.
3. Las llaves contienen objetos.
4. Los corchetes contienen matrices o arreglos

```
{ "empleado":1332,  
  "oficina":"IBM Lima",  
  "telefono":[ { "tipo":"particular", "numero":57380346},  
               { "tipo":"celular", "numero":943789012}  
            ]  
}
```

Tipos de datos

Tipo de dato	Comentario	Ejemplo
\$string	Cadena	"Jorge"
\$int, \$integer, \$long	Entero	1034
\$number	Doble, Flotante	1250.65
\$date	Debe estar en el formato 'yyyy-mm-ttThh:mm:ssZ'	2014-02-18T18:56:00Z
\$timestamp	Timestamp	
\$binary	True, False	
\$oid	Identificador del objeto (Binario)	

Cuando se almacenan documentos JSON para poder ejecutar consultas sobre los documentos se utilizan sentencias NoSQL, que a su vez nos permiten ejecutar operaciones que normalmente se realizan con sentencias SQL. Algunas de las funcionalidades son las siguientes.

Sentencias SQL y funciones	Operadores de documentos JSON
SELECT	\$project
WHERE	\$match
DISTINCT	\$distinct
ORDER BY	\$sort
ROWNUM	\$count

Sentencias SQL y funciones	Operadores de documentos JSON
GROUP BY	\$group
HAVING	\$match
SUM	\$sum
AVERAGE	\$avg
MIN	\$min
MAX	\$max

Con éstos principios podemos comenzar a crear nuestras cadenas de JSON, aunque básicas, ya podríamos interpretarlas y armarlas con cierta facilidad. Algunos editores de código como NetBeans, Dreamweaver, Textpad, Eclipse, etc. tienen incorporados plugins (o se les puede descargar) que nos ayudan a dar formato y validar nuestros objetos JSON.

Herramientas en Internet también los hay como [JSONLint](#), un validador de JSON en línea que nos da formato y valida los objetos que ingresemos en su área de texto.

Por último una nota, cuando se manda o recibe un objeto JSON, siempre se maneja un sólo objeto a la vez, es decir, no podemos manejar más de un objeto pegado con otro aunque los separemos por comas, pero sí es posible enviar/recibir un gran objeto JSON que tenga un valor con muchos objetos en

su interior, por ejemplo si quisiera enviar varios objetos persona de una sola vez, haría lo siguiente:

```
1 {  
2   "personas": [  
3     {  
4       "nombre": "Jean Quiroz",  
5       "edad": 18,  
6       "nacionalidad": "Peruano",  
7       "altura": "165 cm",  
8       "peso": 60,  
9       "pasatiempos": [  
10        "Polo",  
11        "Cricket",  
12        "Ski",  
13        "Drafting",  
14        "Gaming"  
15      ],  
16      "soltero": true,  
17      "direccion": {  
18        "calle": "Ave. Anchoqueta",  
19        "numero": "123",
```

```
20         "pais": "Perú"
21     }
22 },
23 {
24     "nombre": "Peter Yupanqui",
25     "edad": 22,
26     "nacionalidad": "Boliviano",
27     "altura": "178 cm",
28     "peso": 72,
29     "pasatiempos": [
30         "Pintar",
31         "Programar",
32         "Facear"
33     ],
34     "soltero": true,
35     "direccion": {
36         "calle": "Av. Las magnolias",
37         "numero": "123",
38         "pais": "Venezuela"
39     }
40 },
41 {
42     "nombre": "Luisa Wong",
43     "edad": 25,
```

```
44      "nacionalidad": "Japonesa",
45      "altura": "167 cm",
46      "peso": 55,
47      "pasatiempos": [
48          "Natación",
49          "Alpinismo",
50          "Cinéfila",
51          "Socializar",
52          "Gaming"
53      ],
54      "soltero": true,
55      "direccion": {
56          "calle": "Los Conquistadores",
57          "numero": "456",
58          "pais": "Colombia"
59      }
60  }
61 ]
62 }
```

Volviendo al tema de XML...

Para el caso de Chimbote, al hacer el YQL para Chimbote, obtendríamos 418453

Ahora: con el WOEID obtenido podemos consultar por el clima de dicho cualquier lugar, para nuestro caso: De Chimbote

*select * from weather.forecast where woeid=418453*

CODIGO GENERADO POR EL WEB SERVICE YAHOO

```
<?xml version="1.0" encoding="UTF-8"?>
<query xmlns:yahoo="http://www.yahooapis.com/v1/base.rng"
  yahoo:count="1" yahoo:created="2013-12-18T04:40:11Z" yahoo:lang="en-US">
  <diagnostics>
    <publiclyCallable>true</publiclyCallable>
    <url execution-start-time="1" execution-stop-time="6" execution-
time="5"><![CDATA[http://weather.yahooapis.com/forecastrss?w=418453]]></url>
    <user-time>6</user-time>
```

```
<service-time>5</service-time>
<build-version>0.2.2090</build-version>
</diagnostics>
<results>
  <channel>
    <title>Yahoo! Weather - Chimbote, PE</title>
<link>http://us.rd.yahoo.com/dailynews/rss/weather/Chimbote__PE/*http://weather.yahoo
.com/forecast/PEXX0006_f.html</link>
    <description>Yahoo! Weather for Chimbote, PE</description>
    <language>en-us</language>
    <lastBuildDate>Tue, 17 Dec 2013 10:58 pm PET</lastBuildDate>
    <ttl>60</ttl>
    <yweather:location
      xmlns:yweather="http://xml.weather.yahoo.com/ns/rss/1.0"
      city="Chimbote" country="Peru" region=""/>
    <yweather:units
      xmlns:yweather="http://xml.weather.yahoo.com/ns/rss/1.0"
      distance="mi" pressure="in" speed="mph" temperature="F"/>
    <yweather:wind
      xmlns:yweather="http://xml.weather.yahoo.com/ns/rss/1.0"
      chill="66" direction="130" speed="7"/>
    <yweather:atmosphere
      xmlns:yweather="http://xml.weather.yahoo.com/ns/rss/1.0"
      humidity="88" pressure="29.91" rising="0" visibility="3.73"/>
    <yweather:astronomy
      xmlns:yweather="http://xml.weather.yahoo.com/ns/rss/1.0"
      sunrise="5:52 am" sunset="6:30 pm"/>
    <image>
```



```

<title>Yahoo! Weather</title>
<width>142</width>
<height>18</height>
<link>http://weather.yahoo.com</link>
<url>http://l.yimg.com/a/i/brand/purplelogo//uh/us/news-wea.gif</url>
</image>
<item>
  <title>Conditions for Chimbote, PE at 10:58 pm PET</title>
  <geo:lat xmlns:geo="http://www.w3.org/2003/01/geo/wgs84_pos#">-
9.06</geo:lat>
  <geo:long xmlns:geo="http://www.w3.org/2003/01/geo/wgs84_pos#">-
78.58</geo:long>

  <link>http://us.rd.yahoo.com/dailynews/rss/weather/Chimbote__PE/*http://weather.yahoo
.com/forecast/PEXX0006_f.html</link>
  <pubDate>Tue, 17 Dec 2013 10:58 pm PET</pubDate>
  <yweather:condition
    xmlns:yweather="http://xml.weather.yahoo.com/ns/rss/1.0"
    code="31" date="Tue, 17 Dec 2013 10:58 pm PET"
    temp="66" text="Clear"/>
  <description><![CDATA[
<br />
<b>Current Conditions:</b><br />
Clear, 66 F<BR />
<BR /><b>Forecast:</b><BR />
Tue - Partly Cloudy. High: 69 Low: 58<br />
Wed - Partly Cloudy. High: 70 Low: 58<br />
Thu - Partly Cloudy. High: 69 Low: 58<br />
Fri - Partly Cloudy. High: 69 Low: 57<br />

```

```

Sat - Partly Cloudy. High: 69 Low: 57<br />
<br />
<a
href="http://us.rd.yahoo.com/dailynews/rss/weather/Chimbote__PE/*http://weather.yahoo
.com/forecast/PEXX0006_f.html">Full Forecast at Yahoo! Weather</a><BR/><BR/>
(provided by <a href="http://www.weather.com" >The Weather Channel</a><br/>
]]></description>
    <yweather:forecast
      xmlns:yweather="http://xml.weather.yahoo.com/ns/rss/1.0"
      code="29" date="17 Dec 2013" day="Tue" high="69"
      low="58" text="Partly Cloudy"/>
    <yweather:forecast
      xmlns:yweather="http://xml.weather.yahoo.com/ns/rss/1.0"
      code="30" date="18 Dec 2013" day="Wed" high="70"
      low="58" text="Partly Cloudy"/>
    <yweather:forecast
      xmlns:yweather="http://xml.weather.yahoo.com/ns/rss/1.0"
      code="30" date="19 Dec 2013" day="Thu" high="69"
      low="58" text="Partly Cloudy"/>
    <yweather:forecast
      xmlns:yweather="http://xml.weather.yahoo.com/ns/rss/1.0"
      code="30" date="20 Dec 2013" day="Fri" high="69"
      low="57" text="Partly Cloudy"/>
    <yweather:forecast
      xmlns:yweather="http://xml.weather.yahoo.com/ns/rss/1.0"
      code="30" date="21 Dec 2013" day="Sat" high="69"
      low="57" text="Partly Cloudy"/>
    <guid isPermaLink="false">PEXX0006_2013_12_21_7_00_PET</guid>
</item>

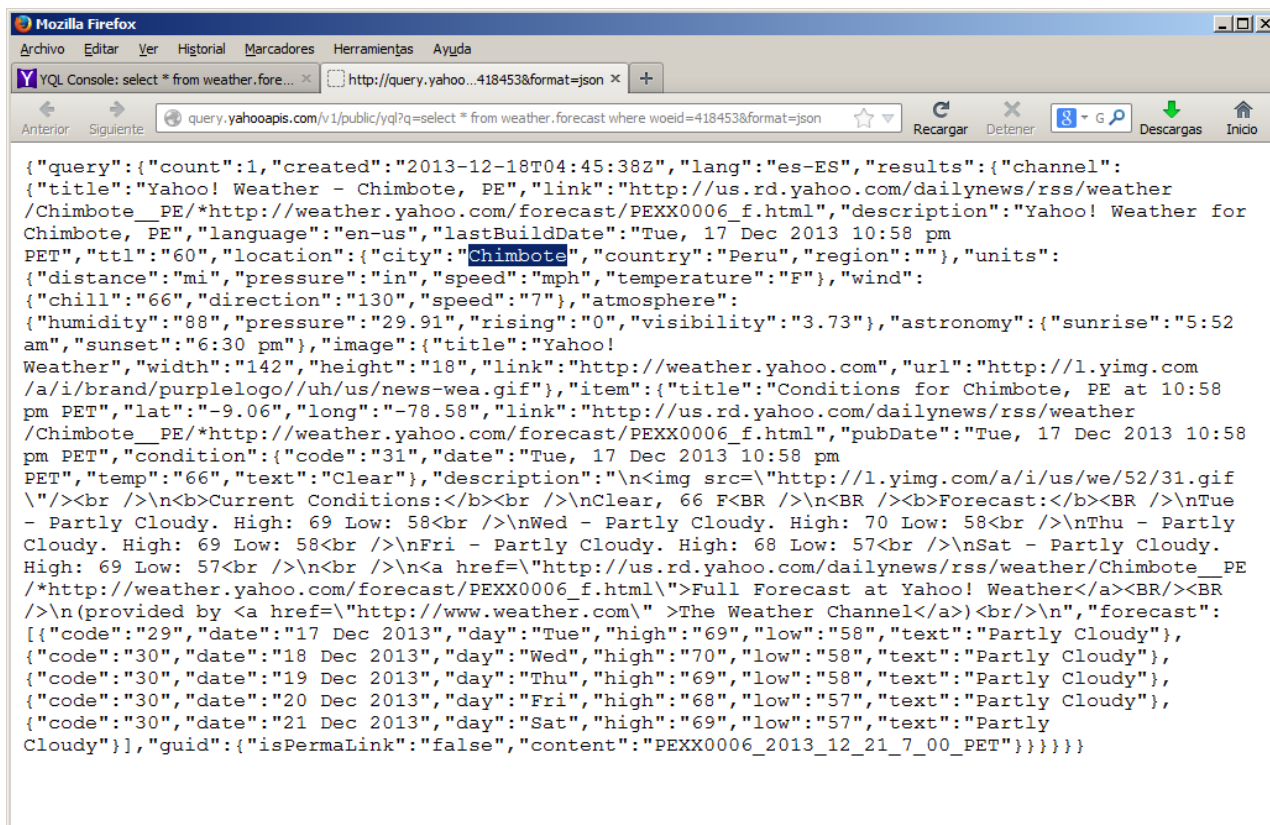
```

```
</channel>  
</results>  
</query>
```

O también podemos hacer uso de web service:

http://query.yahooapis.com/v1/public/yql?q=select%20*%20from%20weather.forecast%20where%20woeid=418453&format=json

<https://weather-ydn-yql.media.yahoo.com/forecastrss?woeid=418453>



Lo anterior lo hicimos solicitando la respuesta en JSON.

Con esta útil herramienta podemos hacer consultas de una serie de lugares, obtener en diversos formatos, solicitar la temperatura en °F o °C, etc.

Se tiene un plugin en:

<http://monkeecreate.github.io/jquery.simpleWeather/>

Visitar:

Nota final: Yahoo Apis ha cambiado de dirección al 2020, razón por la que la urls no pueda funcionar.