

# Desarrollo de Servicios RESTful con Node.js y Oracle Database 12c

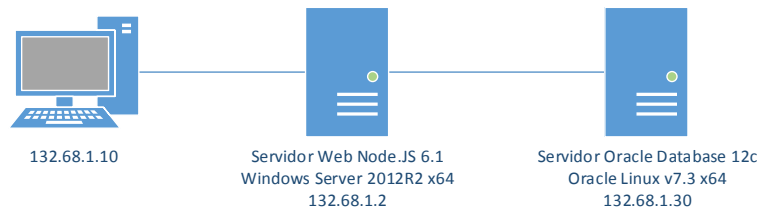
Por Francisco Riccio 

## Introducción

Este artículo está enfocado a explicar cómo podemos implementar servicios RESTful a través Node.js manejando datos ubicados en nuestro Oracle Database 12c; para ello definiré algunos términos con la finalidad de luego utilizarlos en el desarrollo de este artículo.

- Un servicio RESTful es una interfaz que puede interactuar con cualquier dispositivo capaz de comunicarse vía HTTP con la finalidad de obtener o generar operaciones en un repositorio de datos principalmente.
- Node.js es un lenguaje de programación basado en JavaScript el cual se ejecuta en el lado del Servidor y la programación está orientado a eventos permitiendo generar aplicaciones más livianas y eficientes. Este lenguaje se encuentra construido con el motor de JavaScript de Chrome.
- Express es un framework opcional que nos ayudará a reducir nuestro código en el manejo de las operaciones de nuestro servidor web que crearemos.
- bodyParser es una librería que nos permitirá transformar nuestros datos en formato JSON.
- MethodOverride es una librería que nos dará soporte a las operaciones HTTP.

La implementación se desarrolló en una arquitectura multi-capa separando el Servidor Web con la Base de Datos como a continuación se presenta:



Todo el acceso a los datos se realizó utilizando la librería `oracledb` el cual se explicará más adelante.

Al final de la implementación tendremos 4 servicios RESTful implementados en Node.js que ejecutarán operaciones SQL sobre una tabla de Productos alojados en Oracle Database 12c.

Los 4 servicios que se implementarán son:

- Listar todos los productos almacenados.
- Listar el detalle de un producto específico basado en su código.
- Registrar un nuevo producto.
- Eliminar un producto de la base de datos.

## Implementación

La instalación de Node.js 6.1 es simplemente descargar el instalador para el Sistema Operativo correcto y proceder a instalarlo sin mayor dificultad. El Instalador se encuentra en el siguiente url:

<https://nodejs.org/en/download/>

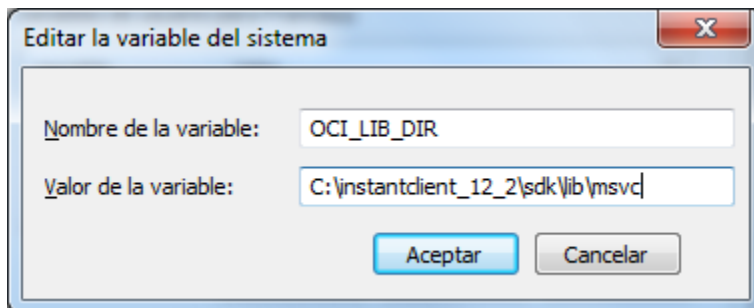
### I. Instalación & Configuración de la Librería oracledb:

La librería oracledb es el driver de conexión a la base de datos Oracle para Node.js. Para instalarlo, el Servidor Web debe cumplir con algunos requisitos los cuales son:

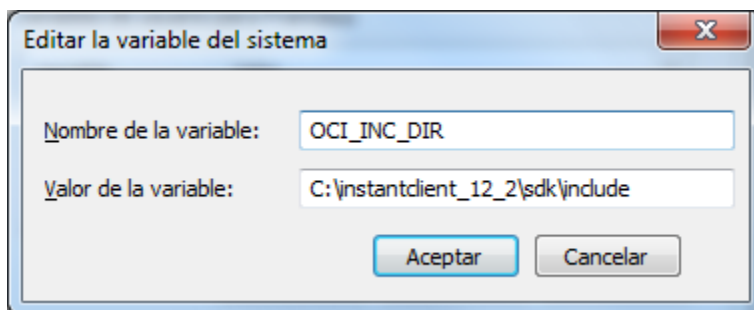
- Instant Oracle Client (**Instant Client Package - Basic + Instant Client Package SDK**). Ambos instaladores deben ser descomprimidos en el mismo directorio y la versión mínima a descargar debe ser 11.2.
- Compilador de C++.
- Python 2.7.

Adicional se debe crear las siguientes variables de ambiente. En mi caso, el InstantClient lo instalé en el directorio C:\instantclient:

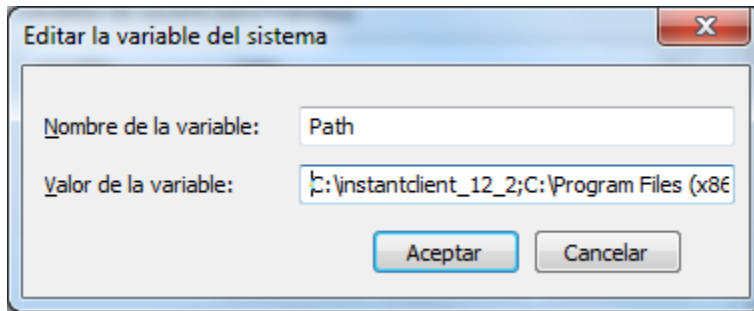
- OCI\_LIB\_DIR



- OCI\_INC\_DIR



- PATH (Agregar el directorio InstantClient y Python).



Una vez que se tenga todos los pre-requisitos creados, procedemos a construir nuestra aplicación que permitirá presentar los servicios RESTful.

## II. Construcción del Proyecto

1. Creamos un usuario y la tabla PRODUCTO en la base de datos como a continuación se presenta:

```
[oracle@srvoracle ~]$ sqlplus system/oracle
SQL*Plus: Release 12.1.0.2.0 Production on Mon May 15 21:55:58 2017
Copyright (c) 1982, 2014, Oracle. All rights reserved.
Last Successful login time: Mon May 15 2017 21:55:18 -05:00

Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application Testing options

SQL> create user friccio identified by oracle;
User created.

SQL> grant connect, resource, unlimited tablespace to friccio;
Grant succeeded.

SQL> connect friccio/oracle
Connected.
SQL> create table PRODUCTO (cod number generated always as identity primary key,
  2 nombre varchar(30), pu number, fecfab date);
Table created.

SQL> insert into PRODUCTO (nombre,pu,fecfab) values ('PRODUCTO1',20,'20-NOV-13');
1 row created.

SQL> commit;
Commit complete.
```

Una vez creada la tabla procedemos con los demás pasos.

2. Creamos una carpeta en el Servidor Windows, en mi caso la he llamado demo.

3. Luego procedemos a cargar las librerías (Express, bodyParser, MethodOverride y oracledb) al proyecto.

Para realizar este punto, debemos crear un archivo llamado package.json donde declararemos las librerías que deseamos instalar, en este caso quedaría así:

```
package.json
{
  "name": "demo",
  "version": "1.0.0",
  "dependencies": {
    "body-parser": "^1.13.2",
    "express": "~4.13.1",
    "method-override": "^2.1.2",
    "oracledb": "^1.13.1"
  }
}
```

Luego ejecutamos el comando **npm install** y tendremos la siguiente salida:

```
C:\demo>npm install
> oracledb@1.13.1 install C:\demo\node_modules\oracledb
> node-gyp rebuild

C:\demo\node_modules\oracledb>if not defined npm_config_node_gyp (node "C:\Program
e_modules\node-gyp\bin\node-gyp.js" rebuild ) else (node "" rebuild )
Los proyectos de esta solución se van a compilar de uno en uno. Para habilitar la
njsOracle.cpp
njsPool.cpp
njsConnection.cpp
njsResultSet.cpp
njsMessages.cpp
njsIntLob.cpp
dpiEnv.cpp
dpiEnvImpl.cpp
dpiException.cpp
dpiExceptionImpl.cpp
dpiConnImpl.cpp
dpiDateTimeArrayImpl.cpp
dpiPoolImpl.cpp
```

4. Creamos un módulo específico que se encargará de conectar a la base de datos y ejecutar las operaciones SQL que necesitemos que se realicen.

Este módulo lo llamaremos dao.js y utilizará la librería oracledb para realizar dicho fin.

```

1 var objoracle = require('oracledb');
2
3 cns = {
4     user: "friccio",
5     password: "oracle",
6     connectString: "132.68.1.30/PRD"
7 };
8
9 function error(err,rs,cn){
10     if (err) {
11         console.log(err.message);
12         rs.contentType('application/json').status(500);
13         rs.send(err.message);
14         if (cn!=null) close(cn);
15         return -1;
16     }
17     else
18         return 0;
19 }
20
21 function open(sql,binds,dml,rs){
22     objoracle.getConnection(cns,function(err,cn){
23         if (error(err,rs,null)==-1) return;
24         cn.execute(sql,binds,{autoCommit: dml},function(err,result){
25             if (error(err,rs,cn)==-1) return;
26             rs.contentType('application/json').status(200);
27             if (dml)
28                 rs.send(JSON.stringify(result.rowsAffected));
29             else{
30                 console.log(result.metaData);
31                 rs.send(JSON.stringify(result.rows));
32             }
33             close(cn);
34         });
35     });
36 }
37
38 function close(cn) {
39     cn.release(
40         function(err) {
41             if (err) { console.error(err.message); }
42         }
43     );
44 }
45
46 exports.open = open;
47 exports.close = close;

```

Explicación del código:

Línea	Explicación
1	Cargamos la librería oracledb y lo referenciamos con la variable objoracle.
3-7	Creamos una variable que almacenará la información de la cadena de conexión de la base de datos Oracle en formato JSON.
9-19	La función error nos permitirá devolver un mensaje de error en formato JSON en caso no se pueda establecer la conexión con la base de datos o cumplir con el requerimiento solicitado por la sentencia SQL.
21	Creamos una función que encapsulará la lógica de conexión a la base de datos, ejecución de la sentencia SQL y finalmente su desconexión.
22-23	Creamos la conexión a la base de datos y de manera asíncrona recibiremos un estado de éxito.
24	En caso de conseguir la conexión con la base de datos, se procede a ejecutar la sentencia SQL requerida. Es importante notar que nuestro código soporta bind variables como parte de la

	<p>sentencia SQL.</p> <p>También se ha especificado que toda sentencia DML se ejecutará con AUTOCOMMIT.</p> <p>Es importante notar que se devolverá el número de registros afectados en caso la sentencia que se desee ejecutar sea una operación DML.</p>
25-32	Se evalúa el éxito de la ejecución de sentencia y en caso de ser exitosa se procede a devolver los datos en formato JSON.
33	Se procede a cerrar la conexión de la base de datos.

La librería `oracledb` puede ser utilizada desde versiones de Node.js 4 y se encuentra disponible en:

<https://github.com/oracle/node-oracledb>

En la siguiente url podemos obtener información completa de todas las funciones disponibles por la librería:

<https://github.com/oracle/node-oracledb/blob/master/doc/api.md>

En ella podemos apreciar que la librería brinda un buen soporte con la base de datos.

Entre sus principales características podemos resaltar las siguientes:

- Soporte a tipos de datos no básicos: `TIMESTAMP`, `TIMESTAMP_TZ`, `BLOB`, `CLOB`, `RAW`.
- Posibilidad de utilizar Bind Variables: `IN`, `OUT`, `IN OUT`.
- Manejo de Pool de Conexiones.
- Posibilidad de utilizar Autenticación Externa.
- Manejo de transacciones.
- Sub-Programas (Stored Procedures & Funciones).
- Soporte a PL/SQL Collection.
- Globalización.
- Client Result Cache.
- Control de FAN (Fast Application Notification).
- Soporte a JSON (Nuevo tipo de dato en la versión Oracle Database 12.1.0.2 y se integra de manera transparente para la extracción y manejo de datos en este formato).

5. Procedemos a crear el módulo principal que se encargará de crear el Servidor Web y presentar los servicios RESTful. Nuestro módulo se llamará app.js.

```
1 var express = require("express");
2 var app = express();
3 var bodyParser = require("body-parser");
4 var methodOverride = require("method-override");
5 var dao = require("./dao");
6
7 app.use(bodyParser.urlencoded({ extended: false }));
8 app.use(bodyParser.json());
9 app.use(methodOverride());
10
11 var router = express.Router();
12
13 router.get('/producto', function(request, response) {
14     var opc = parseInt(request.query.opc);
15     switch (opc) {
16         case 1:
17             sql = "SELECT cod,nombre,pu,feclab FROM producto";
18             dao.open(sql, [], false, response);
19             break;
20         case 2:
21             sql = "SELECT nombre,pu,feclab FROM producto WHERE cod=:cod";
22             var cod = parseInt(request.query.cod);
23             dao.open(sql, [cod], false, response);
24             break;
25         case 3:
26             sql = "INSERT INTO producto(nombre,pu,feclab)" +
27                 " VALUES (:nombre,:pu,TO_DATE(:feclab,'DDMMYYYY'))";
28             var nombre = request.query.nombre;
29             var pu = parseFloat(request.query.pu);
30             var feclab = request.query.feclab;
31             console.log(feclab);
32             dao.open(sql, [nombre,pu,feclab], true, response);
33             break;
34         case 4:
35             sql = "DELETE FROM producto WHERE cod=:cod";
36             var cod = parseFloat(request.query.cod);
37             dao.open(sql, [cod], true, response);
38             break;
39         default:
40             response.contentType('application/json').status(200);
41             response.send(JSON.stringify("Opcion no valida."));
42     }
43     response.end();
44 });
45
46 app.use(router);
47
48 app.listen(3000, function() {
49     console.log("Servidor Web - http://localhost:3000");
50 });
```

Explicación del código:

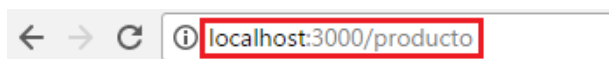
Línea	Explicación
1-5	Cargamos las librerías: express, body-parser, method-override y nuestro módulo dao previamente creado.
7-9, 11,46	Enlazamos algunas funciones y variables a nuestra variable app.
13-44	<p>Recibimos el requerimiento enviado por GET y deberá recibirse una variable llamada "opc" cuyo valor debe estar entre 1 a 4.</p> <p>Si la variable opc tiene el valor de:</p> <ul style="list-style-type: none"> <li>• "1", se consultará todos los productos registrados.</li> <li>• "2", se solicitará el código del producto e irá como una variable bind variable que se construirá como parte de la sentencia SQL para obtener los datos de un producto específico.</li> <li>• "3", se solicitará el código, nombre, precio unitario y fecha de fabricación de un producto para registrarlo.</li> </ul> <p>Como parte de la sentencia SQL utilizamos la función TO_DATE para transformar la cadena de texto que se leerá de la url, cuyo formato será DDMMYYYY (día, mes, año).</p> <ul style="list-style-type: none"> <li>• "4", se solicitará el código del producto para eliminarlo de la base de datos.</li> </ul>
48-50	Se iniciará el Servidor Web publicando los servicios por el puerto 3000 tcp/ip.

6. Iniciar el Servidor Web.

**node app.js**

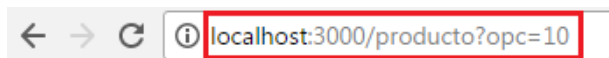
### III. Probar los Servicios RESTful.

1. Probaremos que si no es ingresada la variable opc o su valor no está comprendido entre 1 y 4, se enviará un mensaje indicando que no es una opción válida en formato JSON.



← → ↻ ⓘ localhost:3000/producto

"Opcion no valida."



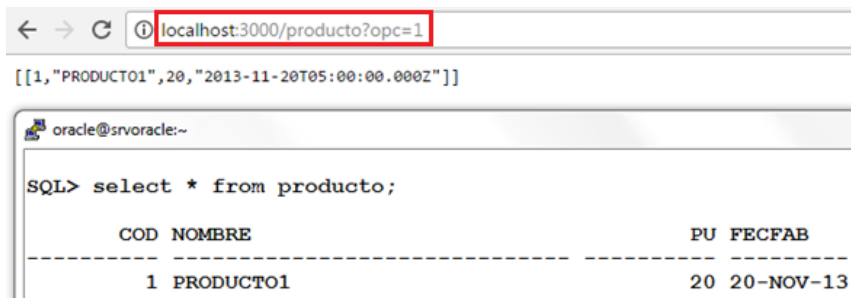
← → ↻ ⓘ localhost:3000/producto?opc=10

"Opcion no valida."



2. Listamos todos los productos registrados.

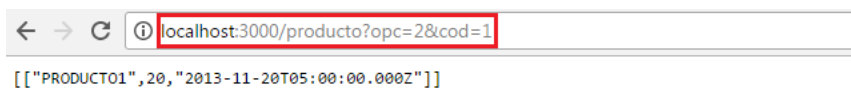
Se adjunta la salida del servicio RESTful y el resultado de la consulta en la base de datos.



```
[[{"COD": "PRODUCTO1", "PU": 20, "FECHFAB": "2013-11-20T05:00:00.000Z"}]]
```

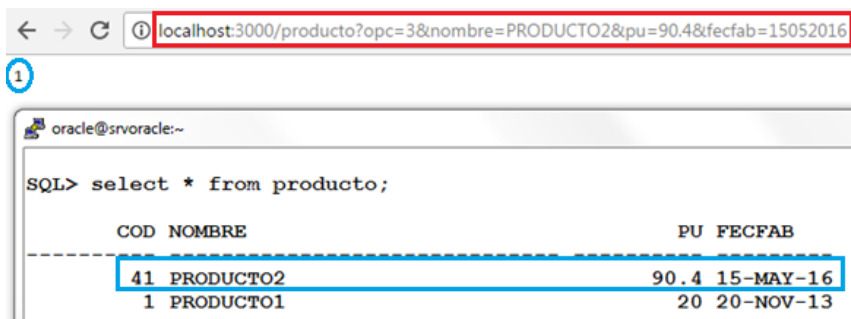
```
oracle@svoracle:~  
SQL> select * from producto;  
  
COD NOMBRE                                PU FECHFAB  
-----  
1 PRODUCTO1                                20 20-NOV-13
```

3. Listamos la información de un producto específico.



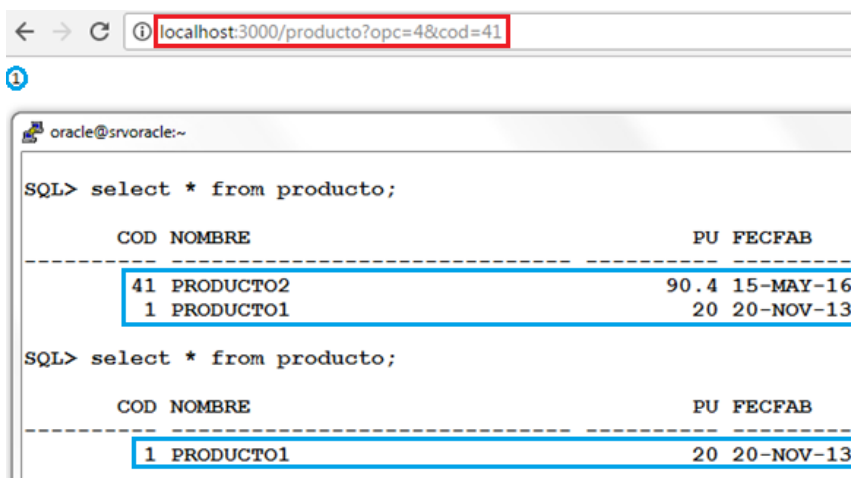
```
[{"COD": "PRODUCTO1", "PU": 20, "FECHFAB": "2013-11-20T05:00:00.000Z"}]]
```

4. Registramos un nuevo Producto.



```
oracle@svoracle:~  
SQL> select * from producto;  
  
COD NOMBRE                                PU FECHFAB  
-----  
41 PRODUCTO2                               90.4 15-MAY-16  
1 PRODUCTO1                                20 20-NOV-13
```

5. Eliminamos el Producto con código 41 de la Base de Datos.



```
oracle@svoracle:~  
SQL> select * from producto;  
  
COD NOMBRE                                PU FECHFAB  
-----  
41 PRODUCTO2                               90.4 15-MAY-16  
1 PRODUCTO1                                20 20-NOV-13  
  
SQL> select * from producto;  
  
COD NOMBRE                                PU FECHFAB  
-----  
1 PRODUCTO1                                20 20-NOV-13
```

Con esta última prueba podemos concluir el trabajo de implementación.

Nuestro Servidor Web en Node.js tendrá la siguiente salida después de ejecutar las pruebas previamente indicadas:

```
Administrador: C:\Windows\system32\cmd.exe - node app.js
C:\demo>node app.js
Servidor Web - http://localhost:3000
[ { name: 'COD' },
  { name: 'NOMBRE' },
  { name: 'PU' },
  { name: 'FECFAB' } ]
[ { name: 'NOMBRE' }, { name: 'PU' }, { name: 'FECFAB' } ]
```

## Conclusión

Oracle ha desarrollado la librería `oracledb` que nos permite trabajar de manera transparente desde Node.js toda la comunicación con la base de datos Oracle incluyendo sus nuevas funcionalidades de la versión 12c como por ejemplo: el uso del tipo de dato JSON, dando la posibilidad de crear aplicaciones ligeras, escalables y con mejor desempeño a través de un Oracle Database 12c en este caso.

Publicado por Ing. Francisco Riccio. Es un Cloud Architect en IBM Perú e instructor de cursos oficiales de certificación Oracle. Está reconocido por Oracle como un Oracle ACE y certificado en productos de Oracle Application & Base de Datos.

e-mail: [franciscoriccio@gmail.com](mailto:franciscoriccio@gmail.com)

web: [www.friccio.com](http://www.friccio.com)