

Universidad de Cartagena

Informe – Web Service

Seminario de actualización

Amaury Rafael Ortega Camargo Y Ramiro Jose Verbel de la Rosa

Contenido

1	Objetivo	2
2	Librerías	2
2.1	javax.jws.WebService	2
2.2	javax.jws.WebMethod	2
2.3	javax.jws.WebParam	3
2.4	javax.jws.Oneway	3
2.5	com.mysql.jdbc.Driver	3
2.6	com.mysql.jdbc.Connection	3
2.7	com.mysql.jdbc.Statement	3
3	Clases	4
3.1	Servidor	4
3.1.1	DB	4
3.1.2	Pc	4
3.1.3	Util	4
3.1.4	WSServidor	5
4	Cliente	6
4.1	Pc	6
4.2	VentanaPrincipal	6
4.2.1	Inicio	6
4.2.2	Detener	7
5	Implementación	7
5.1	Arquitectura	8
5.2	Docker	8
5.2.1	Apache	8
5.2.2	Mysql	8
5.2.3	Resumido	9
5.2.4	Comandos	9
5.2.5	Imagen	9
6	Bibliografía	10

1 Objetivo

Comprender la comunicación de red mediante el uso de Web Services el cual facilita la construcción e implementación de aplicaciones distribuidas basadas en servicios Web HTTP este forma parte de Java EE, usando como mensajes de intercambio JSON (JavaScript Object Notation) el cual es un formato de intercambio de información, con el fin implementar un modelo cliente-servidor el cual proporcione instancias de servidores virtuales que contengan phpMyAdmin y MySQL accesibles mediante red por los clientes.

2 Librerías

Debido a que las librerías para Web Service se encuentran alojadas en Java EE no es necesario añadirla, por el contrario la clase Gson la cual permite el manejo de JSON está alojada en la librería gson-2.8.2.jar, además se hace uso de la librería mysql-connector-java- 5.1.44-bin.jar la cual provee clases que son empleadas para la conexión a la base de datos encargada de registrar los servidores virtuales asignados a los clientes. Por lo tanto, únicamente se presentan en la clase DB logrando el uso de bases de datos MySQL, esta clase se explica posteriormente en el documento.

Específicamente las clases e interfaces usadas en el proyecto son las siguientes:

2.1 javax.jws.WebService

Se usa para definir una clase java como implementación de un servicio web, de esta se llama el método abstracto *serviceName* con el cual se define el nombre del servicio web en nuestro caso *WSServidor*.

```
@WebService(serviceName = "WSServidor")
```

Además se usa como el nombre del servicio WSDL el cual es un formato XML que se utiliza para describir la interfaz publica del servicio web, en este se describe la forma de comunicación, es decir los requisitos del protocolo y los formatos de los mensajes necesarios para interactuar con el Web Service.

2.2 Javax.jws.WebMethod

Se usa para personalizar un método para que sea expuesto como una operación del servicio web, mediante el método abstracto *operationName* con el cual se definen las operaciones que coincidirán en el archivo WSDL, en nuestro caso *iniciarServidor* y *detenerServidor*.

```
@WebMethod(operationName = "iniciarServidor")
```

```
@WebMethod(operationName = "detenerServidor")
```

2.3 Javax.jws.WebParam

Se usa para personalizar la asignación de un parámetro individual a una parte del mensaje del servicio web y un elemento XML, mediante el método abstracto *name* con el cual se define el nombre del parámetro que coincidirá en el archivo WSDL, en nuestro caso el parámetro *id* al momento de detener el servidor.

```
public void detenerServidor @WebParam(name = "id") String id) {
```

2.4 Javax.jws.Oneway

Se usa para Indicar que el @WebMethod dado solo tiene un mensaje de entrada y ningún resultado.

```
@WebMethod(operationName = "detenerServidor")  
@Oneway
```

2.5 com.mysql.jdbc.Driver

Esta clase es llamada para establecer una conexión con la base de datos, solo es instanciada una vez logrado que el driver dentro de la JVM esté listo para ser usado por medio del siguiente código:

```
Class.forName("com.mysql.jdbc.Driver");
```

2.6 com.mysql.jdbc.Connection

Esta clase es usada para instanciar y comenzar una conexión hacia una base de datos, especificando la dirección de red, el puerto y el nombre de la base de datos; además es necesario especificar las credenciales usando usuario y contraseña por medio del siguiente código:

```
Connection conexion = (Connection) DriverManager.getConnection(  
    "jdbc:mysql://" + db_ip + ":" + db_port + "/" + db_name,  
    user, pass);
```

Esta clase también nos permite preparar sentencias SQL destinadas a dicha conexión usando el método `createStatement`.

```
Statement st = (Statement) conexion.createStatement();
```

Al finalizar el uso de dicha conexión, es posible cerrarla usando el método `close()`.

2.7 com.mysql.jdbc.Statement

Finalmente, esta clase nos permite ejecutar las sentencias SQL y en caso necesario retorna un objeto de tipo `ResultSet`. Del objeto es posible obtener el contenido de la DB por medio del nombre de las columnas usando el método `getString(String)`. Esto junto nos permite ejecutar cualquier sentencia en nuestra base de datos por medio de la siguiente estructura:

```
// Saca el ultimo valor de la BD para tener los puertos  
Query = "SELECT * FROM `registros` ORDER BY idUsuario DESC LIMIT 1";  
Statement st = (Statement) conexion.createStatement();  
resultSet = st.executeQuery(Query);  
while (resultSet.next()) {  
    usuario.setPuertoPHP(Integer.parseInt(resultSet.getString("puertoPHP")));  
    usuario.setPuertoSQL(Integer.parseInt(resultSet.getString("puertoSQL")));  
}
```

3 Clases

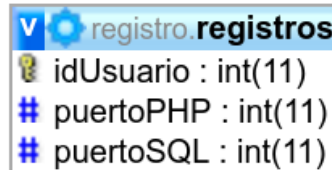
Se implementaron las siguientes clases:

3.1 Servidor

En la aplicación del servidor se tienen las siguientes clases:

3.1.1 DB

Permite manejar la base de datos usando la librería mencionada anteriormente y la clase Pc que será explicada posteriormente. La base datos usada está conformada por la siguiente tabla:



registro.registros	
idUsuario	: int(11)
puertoPHP	: int(11)
puertoSQL	: int(11)

Esta clase nos permite trabajar con la base de datos por los siguientes métodos:

3.1.1.1 *Pc insertar()*

Genera una instancia de la clase Pc la cual contiene el id, puertoPHP y puertoSQL correspondientes al nuevo cliente pidiendo el servicio teniendo en cuenta los registros existentes en la base de datos.

3.1.1.2 *void desconectar()*

Detiene la conexión con la base de datos.

3.1.1.3 *void eliminar(Integer id)*

Elimina el registro en la base datos del usuario con el id proporcionado.

3.1.1.4 *void conectar()*

Crea la conexión con la base de datos.

3.1.2 Pc

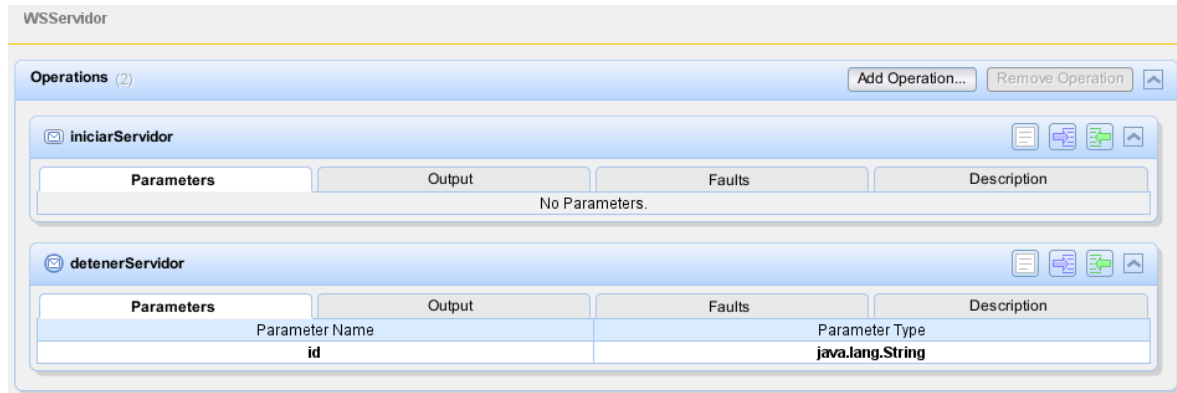
Clase para unificar la información de un cliente del servidor, contiene un id, puertoPHP y puertoSQL. Debido a la sencillez, esta clase solo tiene constructores y getters y setters.

3.1.3 Util

Esta clase tiene como función, almacenar la dirección ip del servidor con la base de datos como string estático y una variable booleana para activar y desactivar el uso de docker.

3.1.4 WSServidor

En esta clase agregamos las operaciones que serán consumidas por los clientes, en el panel de la clase tendremos las operaciones de iniciar y detener el servidor, donde detener servidor recibe un parámetro llamado Id.



En la parte del código se encuentran definidas las operaciones Web de iniciar y detener el servidor, las cuales pueden ser invocadas mediante otra aplicación en nuestro caso el cliente.

Al ejecutar el Web Service se nos da a conocer la URL con la cual es posible aceptar las conexiones de los clientes que necesiten invocar los servicios que este expone; al cliente le es entregado un archivo XML el cual contiene la información necesaria para poder utilizar el Web service, como los nombres de las operaciones y parámetros.

Servicios web

Punto Final		Información	
Nombre de Servicio:	{http://ws/}WSServidor	Dirección:	http://localhost:8080/proyecto-web-service/WSServidor
Nombre de Puerto:	{http://ws/}WSServidorPort	WSDL:	http://localhost:8080/proyecto-web-service/WSServidor?wsdl
		Clase de Implantación:	ws.WSServidor

3.1.4.1 Iniciar Servidor.

La operación de Iniciar Servidor al recibir una petición se crea una conexión a la base de datos usando la clase DB. Inmediatamente se crea un objeto de la clase Pc usando el método insertar() de la clase DB. Este objeto, que llamaremos usuario y nos desconectamos de la base de datos así:

```
DB base_datos = new DB();
base_datos.conectar();
Pc usuario = base_datos.insertar();
base_datos.desconectar();
```

Procedemos a crear el servidor virtual que será dado al cliente usando la información en el objeto usuario. Esto se hace obteniendo un Shell en el sistema operativo del servidor con Docker instalado. Usando una instancia de la clase Runtime, es posible ejecutar comandos con el método exec, esto puede ser almacenado en un objeto de la clase Process lo que nos permitirá esperar que el comando en cuestión haya sido ejecutado usando el método waitFor().

```

Process proceso;
Runtime shell = Runtime.getRuntime();
// COMANDO DOCKER
// docker run -d --rm -p [PuertoPHP]:80 -p [PuertoSQL]:3306 --name=server[ID] xdracklerox/test
proceso = shell.exec("docker run -d --rm -p " + usuario.getPuertoPHP()
    + ":80 -p " + usuario.getPuertoSQL() + ":3306 --name=server"
    + usuario.getId() + " xdracklerox/test");
proceso.waitFor();

```

Luego serializamos el objeto usuario en Json, creado una instancia de Gson e invocamos el método toJson el cual recibe como parámetro el objeto a serializar, y este es dado como respuesta al cliente.

3.1.4.2 Detener Servidor

La operación de Detener Servidor al recibir una petición se procede a crear una conexión a la base de datos usando la clase DB. Esta conexión servirá para eliminar al cliente usando el id obtenido anteriormente usando el método eliminar de la clase DB.

```

DB base_datos = new DB();
base_datos.conectar();
base_datos.eliminar(Integer.parseInt(id));

```

Procedemos a detener y destruir el servidor virtual usando el id obtenido antes. Esto se hace obteniendo un Shell en el sistema operativo del servidor con Docker instalado. Usando una instancia de la clase Runtime, es posible ejecutar comandos con el método exec para detener y destruir el servidor virtual.

```

Process proceso;
Runtime shell = Runtime.getRuntime();
// COMANDO DOCKER
// docker run -d --rm -p [PuertoPHP]:80 -p [PuertoSQL]:3306 --name=server[ID] xdracklerox/test
proceso = shell.exec("docker stop -t 0 server" + id);

```

4 Cliente

4.1 Pc

Clase para unificar la información de un cliente del servidor, contiene un id, puertoPHP y puertoSQL. Debido a la sencillez, esta clase solo tiene constructores y getters y setters.

4.2 VentanaPrincipal

Esta clase es la GUI que usa el cliente para pedir el servicio del servidor, aquí se tienen 2 métodos que corresponden a 2 botones, Iniciar y Detener y además de los métodos para la conexión con el webservice.

4.2.1 Inicio

Se obtiene el Json mediante el método generado por netbeans apartir de las configuraciones realizadas para conectarse con el WSServidor, en pocas palabras consumimos el servicio del Web Service, luego procedemos a deserializar nuestro objeto usuario mediante una instancia de Gson e

invocamos el método `.fromJson` el cual recibe el Json y la clase del objeto y se da a conocer en la ventana.

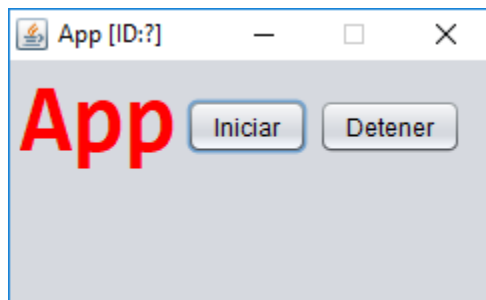
```
// Consumiendo web service
String json = iniciarServidor();
user = new Gson().fromJson(json, Pc.class);
System.out.println("Recibido: " + user);
```

4.2.2 Detener

Al igual que en iniciar se consume la operación detener servidor del web service y se le pasa como parámetro el Id capturado previamente en el inicio, esto hace que se finalice y destruya el contenedor.

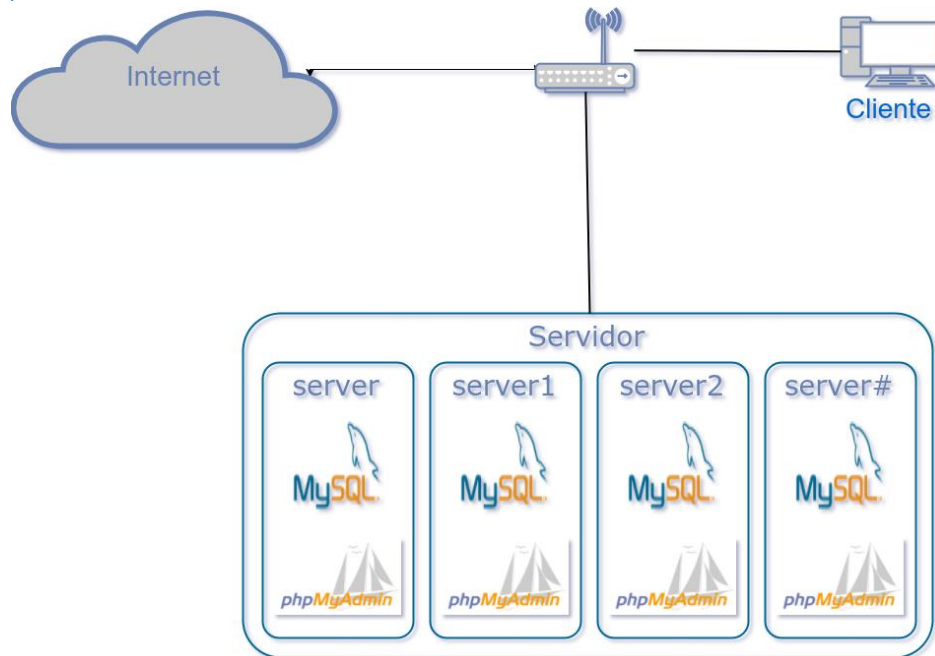
```
// Consumiendo web service
detenerServidor(user.getId());
```

5 Implementación



Pantalla principal cliente

5.1 Arquitectura



5.2 Docker

Docker es una herramienta open-source que nos permite realizar una 'virtualización ligera', con la que poder empaquetar entornos y aplicaciones que posteriormente podremos desplegar en cualquier sistema que disponga de esta tecnología.

Para ello Docker usa LXC (Linux Containers), que es un sistema de virtualización que permite crear múltiples sistemas totalmente aislados entre sí, sobre la misma máquina.

La gran diferencia es que una máquina virtual necesita contener todo el sistema operativo mientras que un contenedor Docker aprovecha el sistema operativo sobre el cual se ejecuta compartiendo el kernel.

Se usó como base la imagen `wnameless/mysql-phpmyadmin:latest` que viene con apache, phpmyadmin pero con configuraciones no funcionales para nuestro proyecto, todo esto bajo Ubuntu 12.04. Los cambios fueron los siguientes:

5.2.1 Apache

Se cambia `/etc/phpmyadmin/apache.conf` añadiendo `allow from all`. Y se reinicia el servicio con `service apache2 restart`. Esto para permitir conexiones fuera de la red P2P que se crea entre el host y el contenedor.

5.2.2 Mysql

Se cambia `/etc/mysql/my.cnf` modificando `bind-address = 127.0.0.1` a `bind-address = *` usando:

```
sed -i "s/.*bind-address.*/bind-address = 0.0.0.0/" /etc/mysql/my.cnf
```

Esto para permitir conexiones fuera de la red P2P que se crea entre el host y el contenedor. Luego se reinicia el servicio con:

```
mysqldadmin shutdown & mysqld_safe
```

Luego, dentro del motor de base de datos se deben dar permisos al usuario root para ser usado por cualquier cliente en la red así:

```
echo "FLUSH privileges;" > sql.sql; echo "CREATE USER 'root'@'%';" >> sql.sql; echo  
"GRANT ALL PRIVILEGES ON * . * TO 'root'@%' WITH GRANT OPTION  
MAX_QUERIES_PER_HOUR 0 MAX_CONNECTIONS_PER_HOUR 0  
MAX_UPDATES_PER_HOUR 0 MAX_USER_CONNECTIONS 0;" >> sql.sql
```

```
mysqld < sql.sql
```

5.2.3 Resumido

Otra forma de hacer lo anterior es con un script creado por nosotros que puede ser usado así:

```
wget https://raw.githubusercontent.com/AmauryOrtega/Sem-  
Update/master/Proyecto-1-Corte/Docker/script.sh && chmod +x script.sh  
&& ./script.sh && rm script.sh
```

5.2.4 Comandos

Es posible trabajar con contenedor de forma manual así:

Iniciar. `docker run -d --rm -p [PuertoPHP]:80 -p [PuertoSQL]:3306 --name=[Nombre] xxdracklerox/test`

Detener. `docker stop -t 0 [Nombre]`

5.2.5 Imagen

Debido a la agrupación de los cambios en script BASH, se pudo simplificar nuestra imagen docker así:

FROM `wnameless/mysql-phpmyadmin:latest`

ADD `script.sh script.sh`

RUN `chmod +x script.sh && ./script.sh && rm script.sh`

Para más información, favor revisar, comentar o hacer peticiones en el repositorio de todo el proyecto (Ortega Camargo & Verbel de la Rosa, Repositorio Sem-Update, 2017) o en el repositorio de aprendizaje de Docker que se usó como fuente de información (Ortega Camargo, Repositorio Learning-Docker, 2017).

6 Bibliografía

Google. (2017). *Repositorio gson*. Obtenido de <https://github.com/google/gson>

Ortega Camargo, A. R. (2017). *Repositorio Learning-Docker*. Obtenido de GitHub.com: <https://github.com/AmauryOrtega/Learning-Docker>

Ortega Camargo, A. R. (s.f.). *Docker Hub*. Obtenido de [xxdrackleroxx/test](https://hub.docker.com/r/xxdrackleroxx/test) PUBLIC REPOSITORY: <https://hub.docker.com/r/xxdrackleroxx/test/>

Ortega Camargo, A. R., & Verbel de la Rosa, R. J. (2017). *Repositorio Sem-Update*. Obtenido de Github: <https://github.com/AmauryOrtega/Sem-Update>

Rodríguez, M. A. (17 de Septiembre de 2012). *Jugando con JSON en Java y la librería Gson*. Obtenido de <https://www.adictosaltrabajo.com/tutoriales/gson-java-json/>

Sanchez, J. (2 de Marzo de 2012). *INGENIERIA DE SISTEMAS Y ELECTRONICA*. Obtenido de Creacion de Un Webservice en Java: <https://ingsistele.wordpress.com/2012/03/02/creacion-de-un-webservice-en-java/>

Systems, B. (2004). *Annotation Type Oneway*. Obtenido de Java EE 5 SDK: <https://docs.oracle.com/javaee/5/api/javax/jws/Oneway.html>

Systems, B. (2004). *Annotation Type WebMethod*. Obtenido de Java EE 5 SDK: [https://docs.oracle.com/javaee/5/api/javax/jws/WebMethod.html#operationName\(\)](https://docs.oracle.com/javaee/5/api/javax/jws/WebMethod.html#operationName())

Systems, B. (2004). *Annotation Type WebParam*. Obtenido de Java EE 6: [https://docs.oracle.com/javaee/6/api/javax/jws/WebParam.html#name\(\)](https://docs.oracle.com/javaee/6/api/javax/jws/WebParam.html#name())

Systems, B. (2004). *Annotation Type WebService*. Obtenido de Java EE 5 SDK: [https://docs.oracle.com/javaee/5/api/javax/jws/WebService.html#serviceName\(\)](https://docs.oracle.com/javaee/5/api/javax/jws/WebService.html#serviceName())