Using JMX to Monitor Oracle Weblogic Partitions

1. Introduction

In this document a monitor is created using JMX. This monitor is made to show the performance of a Data Source, which is deployed on a partition. Thus, MBeans are used not only to discover managed servers, but also to show partitions. In addition, as the main goal is to show the connection pool's runtime state (do not forget data sources belong to partitions) MBeans are also used to discover the resource groups related to each partition. Furthermore, the program includes a class that generates JDBC connection leaks to demonstrate how to monitor this kind of problems.

In order to avoid any confusion related to matters such as partitions, MBeans, JMX and JDBC, some concepts are shown in the following section. After that, the code source of the program is explained in detail to conclude with some screens of the GUI developed.

2. Concepts

a. Java Management Extension (JMX). It is a framework that was created by Sun in order to manage applications and devices. One of the main advantages of JMX is its capacity of going through different layers of an architecture without modifying the original code. With this in mind, JMX does not have problems to work on different operating systems and networking protocols (Jiang, H. et al., 2010). This definition given by Jiang, H et al. (2010) makes sense since JMX is developed on Java so it leverages the language portability.

b. MBean. The previous point states that JMX is a framework to administer applications and devices. Thus, it is necessary a component, which exposes the properties and states of applications, devices and in this case application servers. This component is called MBean and in addition, it allows the reconfiguration of application server's properties (Mazanatti N. et al., 2013)

c. Domain. A domain is a logical structure that includes machines, servers, clusters, resources, etc. It means a domain includes every component from a Weblogic Server. According to Schildmeijer, M (2011) a domain is the most important administrative unit and at least has an Administration Server.

d. Partition. This is one of the new concepts introduced by Oracle in Weblogic 12.2.1 as part of the multitenant architecture. A partition is a kind of micro container that allows splitting a domain into several independent parts called partitions. Of course, a partition is only a part from the new architecture, which includes other matters such as virtual targets, resource groups, etc. According to Oracle (2016) as cited by Castillo R. (2016) *"…a partition is an administrative and runtime unit that is equivalent to a portion of a domain, which is used to run applications and their resources."*

e. **Virtual targets.** A virtual target defines two important things, a target pointed by resource groups at the partition or domain level and a HTTP server used by each target (Oracle, 2016) as cited by Castillo R. (2016).

f. **Resource group.** A resource group is a logical way to organize resources and applications according to their use, environment or any criteria defined by a business. For example, it is possible to organize environments such as development and testing using resource groups to isolate applications and resources. If you have several modules such as orders, financial risk, fulfilments and so on, resource groups allows you to organize applications and resources that before were scattered.

g. **Data source and connection pools.** Data sources are interfaces between applications and databases that allow developers to forget about database's technical details. Data sources are implemented using Java Database Connectivity (JDBC) and defines a dynamic set of connections to the database. In addition, data sources are used by applications through Java Naming and Directory Interface (JNDI) (Schildmeijer, M, 2011).

   Connection pools are sets of connections to the database that could be increased and decreased dynamically. Since connecting applications with the database is an expensive operation, connection pools represents an advantage because they can be reused avoiding the connections creation on demand (Schildmeijer, M, 2011).

h. **JDBC connection leaks.** After a connection from the connection pool is used by an application, this connection (logical connection) is closed and then the connection (the physical one) is returned to the connection pool to be reused. However, when an application does not implement the close method properly, physical connections are not released. Thus, over the time the connection pool will not have physical connections to manage the applications demands. This problem is known as JDBC connection leak and could cause the crash of managed servers (Castillo, R., 2015). Therefore, giving ways to monitor this problem is a relevant matter.

i. **Topology used,** the topology used in this post is depicted by figure 1.

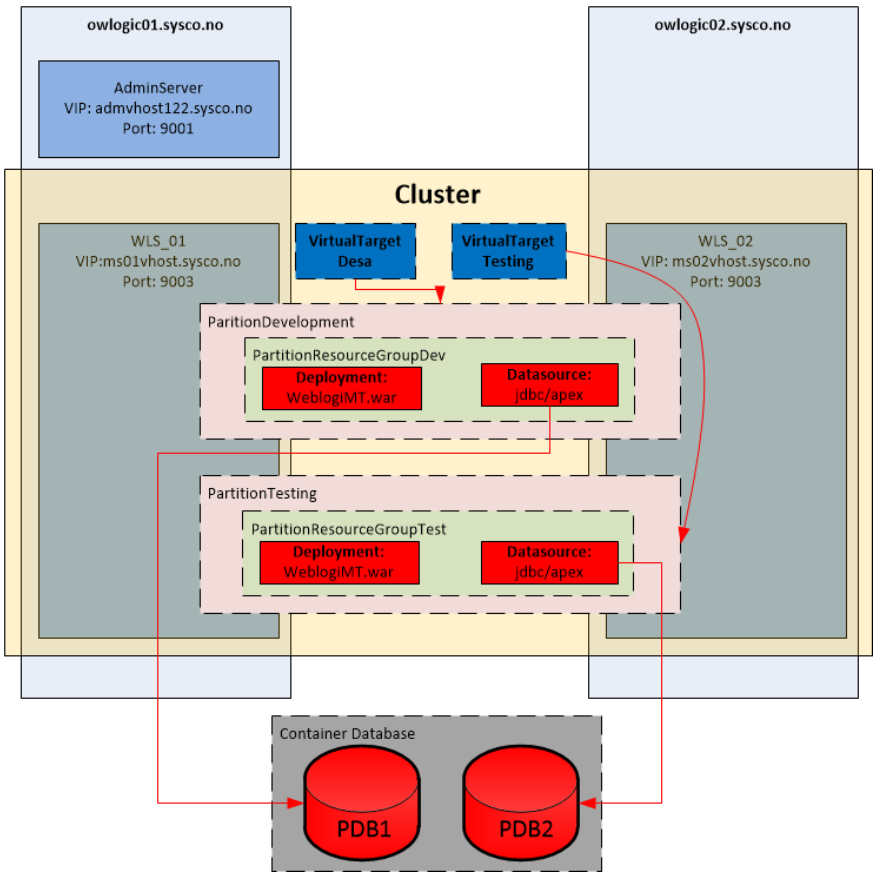


Figure 1. Source: Castillo, R (2016)

3. Source code explanation

The program developed in this post establishes a connection with the Weblogic server domain to show severs, partitions and data sources that belong to those partitions. In addition, data sources detected are used to generate a JDBC connection leak. This is the GUI used in this demonstration.

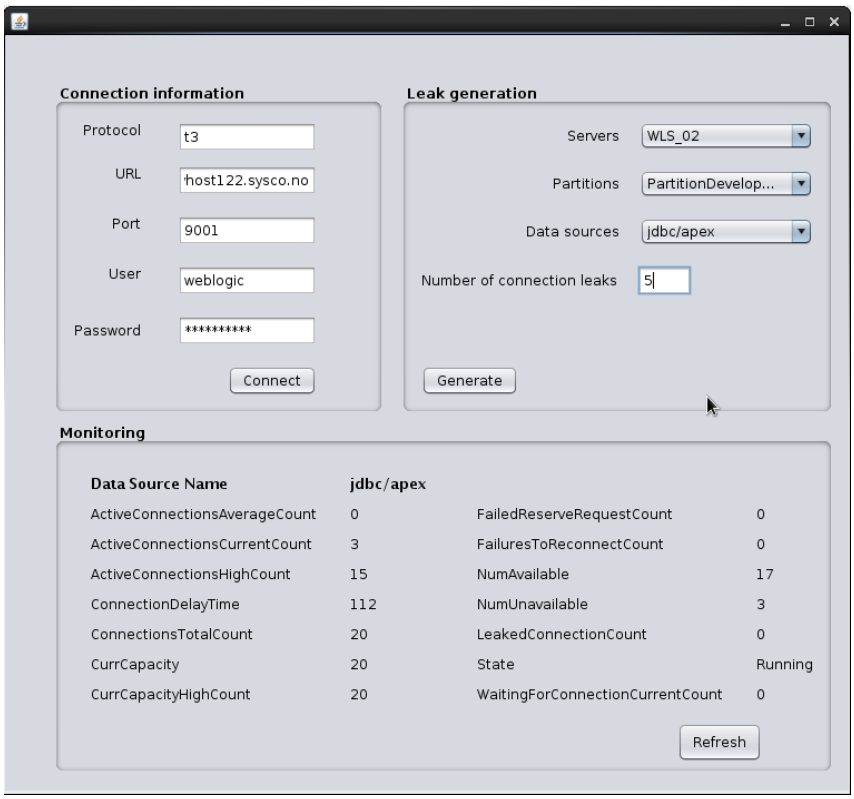


Figure 2. GUI created for this demonstration

The GUI shown in figure 2 shows three sections

- **Connection information**. In this section parameters used to connect to the Weblogic server such as protocol, URL, port, user and password are given by the user. After clicking the "Connect" button, the "Leak Generation" section is filled.

- **Leak Generation**. This section allows users to select severs partitions and the data source that belongs to this combination of servers and partitions. In addition, here it is possible to set the number of connection leaks user wants to generate.

- **Monitoring**. Using the button "Refresh" it is possible to monitor the main indicators that belong to the data source under investigation. For example, in figure 2 it is possible to see the state of data source "jdbc/Apex" is equal to Running.

Following each class is explained.

Class JMXConnection

- **Definition**: Class used to establish the connection with the weblogic domain.

```
public class JMXConnection {
    private MBeanServerConnection connection;
    private String domainName;
    private JMXConnector connector;
    private MBeanServerConnection connectionConfig;
    private JMXConnector connectorConfig;
    private String user;
    private String password;
    private String hostname;
    private int port;
    private String protocol;
```

- **Constructor**: This class has a constructor that receives and initializes all the connection parameters needed to establish a connection with the Weblogic domain.

```
public JMXConnection(String user, String password, String hostname, String port, String protocol){

    this.user=user;
    this.password=password;
    this.hostname=hostname;
    this.port=Integer.valueOf(port);
    this.protocol=protocol;
}
```

- **Methods**
  - **public MBeanServerConnection getConnection()**
    This method establishes the connection with the Weblogic domain and returns an MBeanServerConnection object that is used during the program execution to access to Weblogic servers (Administrator and managed servers) and other resources.

```
public MBeanServerConnection getConnection() {
    String jndiroot = "/jndi/";
    String runTime = "weblogic.management.mbeanservers.domainruntime";
    HashMap h = new HashMap();
    h.put(Context.SECURITY_PRINCIPAL, this.user);
    h.put(Context.SECURITY_CREDENTIALS, this.password);
    h.put(JMXConnectorFactory.PROTOCOL_PROVIDER_PACKAGES,"weblogic.management.remote");

    try
    {
      JMXServiceURL serviceURL = new JMXServiceURL(this.protocol, this.hostname, this.port, jndiroot + runTime);
      connector = JMXConnectorFactory.connect(serviceURL, h);
      connection = connector.getMBeanServerConnection();
      domainName=connection.getDefaultDomain();
    }
    catch(Exception e)
    {
      e.printStackTrace();
    }
   /* finally{
        try{
            connector.close();
        }
        catch(Exception e)
        {
          e.printStackTrace();
        }
    }*/
    return connection;

}
```

It is important to remark that in this case the string "weblogic.management.mbeanservers.domainruntime" is used. It is the same as using this command on WLST.

```
wls:/chavin/serverConfig/> domainRuntime()
Location changed to domainRuntime tree. This is a read-only tree
with DomainMBean as the root MBean.
For more help, use help('domainRuntime')

wls:/chavin/domainRuntime/>
```

Class ServerMonitoring

- **Definition**: Class used to get the list of servers (Admin and managed) in a domain.

```
public class ServerMonitoring {

    private ObjectName listServers[];
    private static String combea = "com.bea:Name=";
    private static String service = "DomainRuntimeService,Type=weblogic.management.mbeanservers.domainruntime.DomainRuntimeServiceMBean";

    public ObjectName [] getListServers(MBeanServerConnection connection){
```

The MBean defined by this string:

"com.bea:Name=DomainRuntimeService,Type=weblogic.management.mbeanservers
.domainruntime.DomainRuntimeServiceMBean"

Could be also analysed using WLST as is shown in this picture.

```
wls:/chavin/domainRuntime/DomainServices/DomainRuntimeService> cd('ServerRuntimes')
wls:/chavin/domainRuntime/DomainServices/DomainRuntimeService/ServerRuntimes> ls()
dr--   AdminServer
dr--   WLS_01
dr--   WLS_02

wls:/chavin/domainRuntime/DomainServices/DomainRuntimeService/ServerRuntimes> cmo
com.bea:Name=DomainRuntimeService,Type=weblogic.management.mbeanservers.domainruntime.DomainRuntimeServiceMBean
wls:/chavin/domainRuntime/DomainServices/DomainRuntimeService/ServerRuntimes>
```

- **Methods**
  - **public ObjectName [] getListServers(MBeanServerConnection connection)**
    This method receives the connection object in order to get the list of server. To do this, the method used the MBean defined by the concatenation of variables called *combea* and *service.*

```
public ObjectName [] getListServers(MBeanServerConnection connection){
    try{
    ObjectName temp =new ObjectName(combea + service);

    listServers =(ObjectName[])connection.getAttribute(temp,"ServerRuntimes");

    }
    catch (Exception e){
        e.printStackTrace();
    }
    return listServers;
}
```

Class PartitionMonitoring

- **Definition**: Class used to get the list of partitions related to a server.

```
package monitoring;

import javax.management.MBeanServerConnection;
import javax.management.ObjectName;

/**
 *
 * @author oracle
 */
public class PartitionMonitoring {
    private ObjectName[] listPartitions;
```

- **Methods**
  - **Public ObjectName [] getListPartition (String serverName, MBeanServerConnection connection)**
    This method receives the server name and returns the list of partitions, which belong to the server.

```
public ObjectName [] getListPartitions(String serverName, MBeanServerConnection connection){
    try{
        //com.bea:Name=WLS_01,Location=WLS_01,Type=ServerRuntime
       String service="com.bea:Name="+serverName+",Location="+serverName+",Type=ServerRuntime";
       //String service="com.bea:Name="+"WLS_01"+",Location="+"WLS_01"+",Type=ServerRuntime";
        ObjectName temp =new ObjectName(service);

        listPartitions=  (ObjectName[])connection.getAttribute(temp,"PartitionRuntimes");
    }
    catch (Exception e){
        e.printStackTrace();
    }
    return listPartitions;
}
```

This is the MBean used opened from WLST

```
wls:/chavin/domainRuntime/ServerRuntimes/WLS_01/PartitionRuntimes> cmo
[MBeanServerInvocationHandler]com.bea:Name=WLS_01,Location=WLS_01,Type=ServerRuntime
wls:/chavin/domainRuntime/ServerRuntimes/WLS_01/PartitionRuntimes> ls()
dr--   PartitionDevelopment
dr--   PartitionTesting
```

Class JDBCMonitoring

- **Definition**: Class used to get the list of data sources per server and partition and to show the main runtime indicators that belong to the data source.

```
public class JDBCMonitoring {
    private ObjectName[] listDataSources;
    private String[] JNDINames;

    private String nameDS;
    private String ActiveConnectionsAverageCount;
    private String ActiveConnectionsCurrentCount;
    private String ActiveConnectionsHighCount;
    private String ConnectionDelayTime;
    private String ConnectionsTotalCount;
    private String CurrCapacity;
    private String CurrCapacityHighCount;

    private String FailedReserveRequestCount;
    private String FailuresToReconnectCount;
    private String NumAvailable;
    private String NumUnavailable;
    private String LeakedConnectionCount;
    private String State;
    private String WaitingForConnectionCurrentCount;
```

- **Methods**
  - **public void refreshAttributes(MBeanServerConnection connection, ObjectName dataSourceName)**

    This method receives the connection object and the data source name to read its runtime indicators.

```
public void refreshAttributes(MBeanServerConnection connection, ObjectName dataSourceName){
    try {
        nameDS=connection.getAttribute(dataSourceName, "Name").toString();
        ActiveConnectionsAverageCount=connection.getAttribute(dataSourceName,"ActiveConnectionsAverageCount").toString();
        ActiveConnectionsCurrentCount=connection.getAttribute(dataSourceName,"ActiveConnectionsCurrentCount").toString();
        ActiveConnectionsHighCount=connection.getAttribute(dataSourceName,"ActiveConnectionsHighCount").toString();
        ConnectionDelayTime=connection.getAttribute(dataSourceName,"ConnectionDelayTime").toString();
        ConnectionsTotalCount=connection.getAttribute(dataSourceName,"ConnectionsTotalCount").toString();
        CurrCapacity=connection.getAttribute(dataSourceName,"CurrCapacity").toString();
        CurrCapacityHighCount=connection.getAttribute(dataSourceName,"CurrCapacityHighCount").toString();
        FailedReserveRequestCount=connection.getAttribute(dataSourceName,"FailedReserveRequestCount").toString();
        FailuresToReconnectCount=connection.getAttribute(dataSourceName,"FailuresToReconnectCount").toString();
        NumAvailable=connection.getAttribute(dataSourceName,"NumAvailable").toString();
        NumUnavailable=connection.getAttribute(dataSourceName,"NumUnavailable").toString();
        LeakedConnectionCount=connection.getAttribute(dataSourceName,"LeakedConnectionCount").toString();
        State=connection.getAttribute(dataSourceName,"State").toString();
        WaitingForConnectionCurrentCount=connection.getAttribute(dataSourceName,"WaitingForConnectionCurrentCount").toString();
    }
    catch (Exception e){
        e.printStackTrace();
    }
}
```

  - **public ObjectName [] getListDS(String serverName, MBeanServerConnection connection)**

    This method receives the server name and the connection to the Weblogic server to get the list of data sources that belong to the domain.

```
//To get Data Sources that belongs to the Domain
public ObjectName [] getListDS(String serverName, MBeanServerConnection connection){
    try{
        String service="com.bea:ServerRuntime="+serverName+
                ",Name="+serverName+",Location="+serverName+
                ",Type=JDBCServiceRuntime";
        //ObjectName temp =new ObjectName("com.bea:ServerRuntime="+serverName+",Name="+serverName+",L
        ObjectName temp =new ObjectName(service);

        listDataSources=  (ObjectName[])connection.getAttribute(temp,"JDBCDataSourceRuntimeMBeans");
    }
    catch (Exception e){
        e.printStackTrace();
    }
    return listDataSources;
}
```

    This is the MBean used opened from WLST.

```
wls:/chavin/domainRuntime/ServerRuntimes/WLS_01/JDBCServiceRuntime/WLS_01/JDBCDataSourceRuntimeMBeans> cmo
[MBeanServerInvocationHandler]com.bea:Name=WLS_01,ServerRuntime=WLS_01,Location=WLS_01,Type=JDBCServiceRuntime
wls:/chavin/domainRuntime/ServerRuntimes/WLS_01/JDBCServiceRuntime/WLS_01/JDBCDataSourceRuntimeMBeans>
```

- **public ObjectName [] getListDS(String serverName, String partitionName, MBeanServerConnection connection)**

This method receives the server name, the partition name and the connection to the Weblogic server to get the list of data sources that belong to the partition.

```
//To get Data Sources that belongs to a Partition
public ObjectName [] getListDS(String serverName, String partitionName, MBeanServerConnection connection){
    try{
        String service="com.bea:Name="+partitionName+",ServerRuntime="+
                serverName+",Location="+serverName+
                ",Type=JDBCPartitionRuntime"+",PartitionRuntime="+partitionName;

        //com.bea:Name=PartitionDevelopment,ServerRuntime=WLS_01,Location=WLS_01,Type=JDBCPartitionRuntime,

        ObjectName temp =new ObjectName(service);

        listDataSources=  (ObjectName[])connection.getAttribute(temp,"JDBCDataSourceRuntimeMBeans");
    }
    catch (Exception e){
        e.printStackTrace();
    }
    return listDataSources;
}
```

This is the MBean used opened from WLST

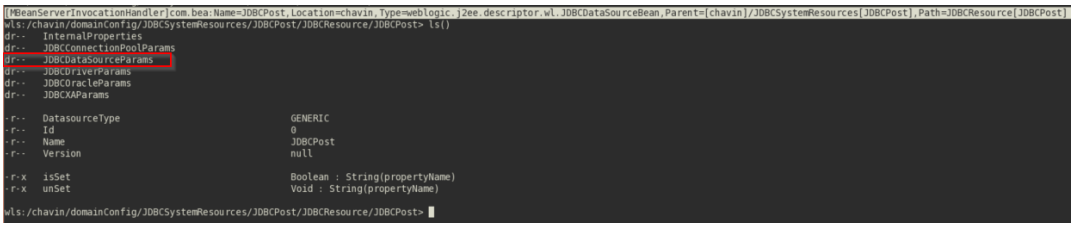
```
wls:/chavin/domainRuntime/ServerRuntimes/WLS_01/PartitionRuntimes/PartitionDevelopment/JDBCPartitionRuntime/PartitionDevelopment/JDBCDataSourceRuntimeMBeans> ls()
dr--   jdbc/apex

wls:/chavin/domainRuntime/ServerRuntimes/WLS_01/PartitionRuntimes/PartitionDevelopment/JDBCPartitionRuntime/PartitionDevelopment/JDBCDataSourceRuntimeMBeans> cmo
[MBeanServerInvocationHandler]com.bea:Name=PartitionDevelopment,ServerRuntime=WLS_01,Location=WLS_01,Type=JDBCPartitionRuntime,PartitionRuntime=PartitionDevelopment
wls:/chavin/domainRuntime/ServerRuntimes/WLS_01/PartitionRuntimes/PartitionDevelopment/JDBCPartitionRuntime/PartitionDevelopment/JDBCDataSourceRuntimeMBeans>
```

- **public String getJNDINames(String dataSourceName, String domainName, MBeanServerConnection connection)**

As this program is also used to generate JDBC connection leaks, it is necessary to know the JNDI name associated to each data source in order to get a connection. This method gets the JNDI name based on the data source name, the domain name and the connection object. Of course, the user does not need to enter the domain name.

```
//JNDI names dor the Domain - WLST domainConfig() branch
public String getJNDINames(String dataSourceName, String domainName, MBeanServerConnection connection){
    try{
        String service="com.bea:Name="+dataSourceName+",Location="+domainName+
                ",Type=weblogic.j2ee.descriptor.wl.JDBCDataSourceParamsBean,Parent=["+
                        domainName+"]/JDBCSystemResources["+dataSourceName+
                "],Path=JDBCResource["+dataSourceName+"]/JDBCDataSourceParams";

        //ObjectName temp =new ObjectName("com.bea:ServerRuntime="+serverName+",Name="+serverName+",Loca

        ObjectName temp =new ObjectName(service);

        JNDINames=  (String[])connection.getAttribute(temp,"JNDINames");
    }
    catch (Exception e){
        e.printStackTrace();
    }
    return JNDINames[0];
}
```

This is the MBean used opened from WLST



- **public String getJNDINames(String dataSourceName, String domainName, String partitionName, MBeanServerConnection connection)**

  This method gets the JNDI name based on the data source name, the domain name, the partition name and the connection object. Of course, the user does not need to enter the domain name.

```
public String getJNDINames(String dataSourceName, String domainName, String partitionName, MBeanServerConnection connection){
    try{

        String servTmp="com.bea:Name="+partitionName+",Location="+domainName+",Type=Partition";
        ObjectName objtmp =new ObjectName(servTmp);
        ObjectName[] resourceGroups=  (ObjectName[])connection.getAttribute(objtmp,"ResourceGroups");


        /*At this date to keep it simple, it is assumed there is only a Resource Group within each partition.
        In the future the possibility of having more RG will be implemented
        */

        String resourceGroupName = (String) connection.getAttribute(resourceGroups[0],"Name");


        String service="com.bea:Name="+dataSourceName+",Location="+domainName+
                ",Type=weblogic.j2ee.descriptor.wl.JDBCDataSourceParamsBean,Parent=["+
                        domainName+"]/Partitions["+partitionName+"]/ResourceGroups["+resourceGroupName+
                "]/JDBCSystemResources["+dataSourceName+"],Path=JDBCResource["+dataSourceName+"]/JDBCDataSourceParams";

        System.out.println("Service: " + service);


        ObjectName temp =new ObjectName(service);

        JNDINames=  (String[])connection.getAttribute(temp,"JNDINames");
    }
    catch (Exception e){
        e.printStackTrace();
    }

    System.out.println("getJNDINames " + JNDINames[0]);

    return JNDINames[0];
}
```

This is the MBean used opened from WLST

Class DBConnection

- **Definition**: Class used to generate JDBC connection leaks on a data source that is received as a parameter.

```
public class DBConnection {
    private Hashtable ht=new Hashtable();
    private Connection conn = null;
    private Statement stmt = null;
    private ResultSet rs = null;
    private Context ctx = null;
```

- **Constructor:** The constructor receives the provider's URL to establish the connection with the database. In the case of partitions the provider's URL has this format:

  t3://ms01vhost122.sysco.no:9003/partitions/PartitionDevelopment

```
public DBConnection(String urlProvider){
    ht.put(Context.INITIAL_CONTEXT_FACTORY,
     "weblogic.jndi.WLInitialContextFactory");
     ht.put(Context.PROVIDER_URL,
     urlProvider);
    /*ht.put(Context.SECURITY_PRINCIPAL,"admin
    ht.put(Context.SECURITY_CREDENTIALS,"weblo
}
```

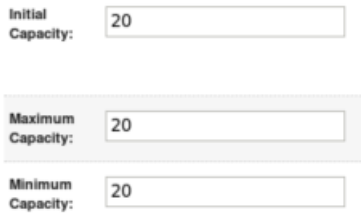
- **Methods**
  - **public void establishConnection(String dataSourceName)**
    This method receives the dataSourceName and stablishes the connection against the database. Connections are not closed on purpose to generate the JDBC connection leak.

```
public void establishConnection(String dataSourceName){
    try {

        System.out.println("establishConnection method - parameter dataSourceName: " + dataSourceName);


        ctx = new InitialContext(ht);
        javax.sql.DataSource ds = (javax.sql.DataSource) ctx.lookup(dataSourceName);

        conn = ds.getConnection();
       // You can now use the conn object to create
       //  Statements and retrieve resulrt sets:
        stmt = conn.createStatement();
        //stmt.execute("select * from dual");
        //rs = stmt.getResultSet();

    //Close JDBC objects as soon as possible
        //stmt.close();
        //stmt=null;
        //conn.close();
        //conn=null;
     }
    catch (Exception e) {
        e.printStackTrace();
      // a failure occurred
      //log message;

        //ctx.close();
        //if (rs != null) rs.close();
        //if (stmt != null) stmt.close();
        //if (conn != null) conn.close();
    }
```

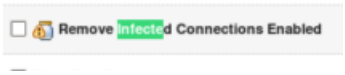
4. Demonstration

In this demonstration, the application is used to generate JDBC connection leaks on the Weblogic partition. First, the connection pool configuration is detailed.

The initial capacity, maximum capacity and minimum capacity of data source called **jdbc/Apex** is set on 20.



The property remove infected connection is disabled. Currently it is enabled by default.



The parameter Inactive Connection Timeout is set on 30s. It is too short, but it is useful for a demonstration.

Now it is time to execute the program.

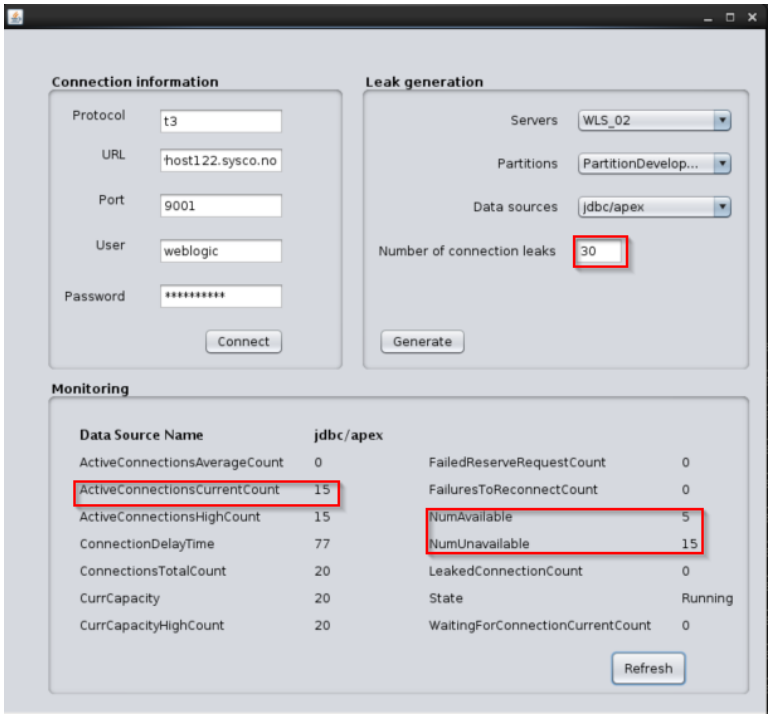


Figure 3

As can be seen in figure 3 even though 30 connections were used and the capacity of the data source is equal to 20, there are 15 busy connections and 5 available connections so the question is why? It happens because the provider URL used to connect to the database includes the partition information, i.e. ***t3://ms01vhost122.sysco.no:9003/partitions/PartitionDevelopment***

and the partition uses a virtual target that is deployed on a cluster. Thus, the connections are balanced between two servers (15 connections per server) that are part of the cluster.

In addition, figure 4 shows the connections were released after 30 seconds to prevent the connection leak.

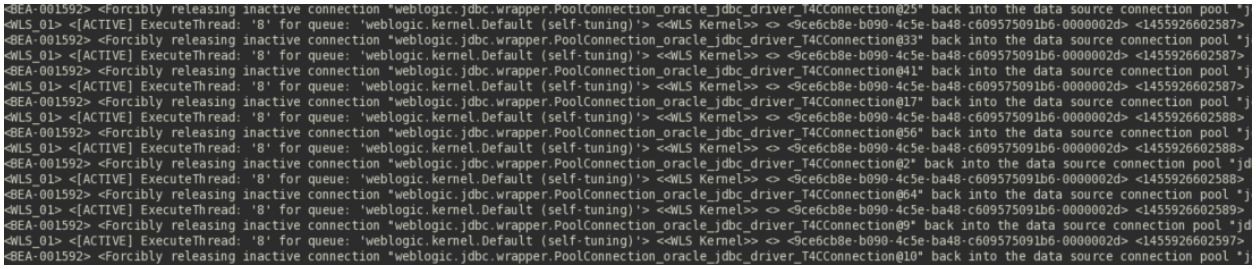


Figure 4

However, there is a problem because the number of connections leaks detected by our programs was not updated as is shown in figure 6.

Figure 6

Does the program have a problem? It seems to be the answer is not because even though the log file shows the server has had released connections, the administrative console does not show any evidence about the number of leak connections as can be seen in figure 7.

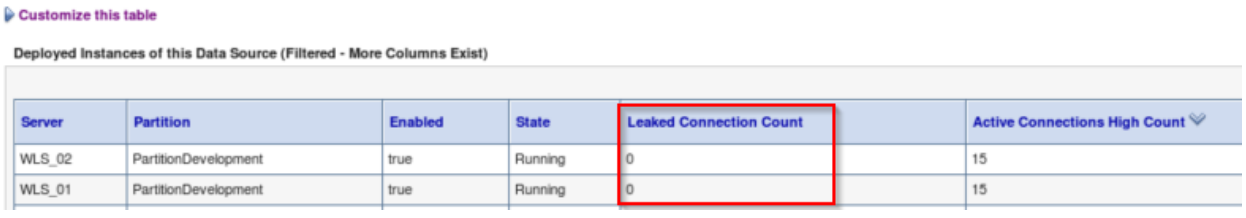


Figure 7

Conclusions. This post has shown how to connect to the Weblogic server using some Java classes. It is important to remark that this demonstration could be used as a base to generate more classes in order to monitor more server's components. It would be interesting for customers who do not have any kind of monitoring system at the application server level. What is more, partitions were used demonstrating that using JMX on partitions is a simple process.

Last but not least, it seems to be there is a problem with the MBean that shows the information about connection leaks because it did not work as is expected.

5. References list

Jiang, H, Hai, L, Wang, N, & Di, R 2010, 'A performance monitoring solution for distributed application system based on JMX', *Proceedings - 9Th International Conference On Grid And Cloud Computing, GCC 2010*, Proceedings - 9th International Conference on Grid and Cloud Computing, GCC 2010, p. 124-127, Scopus®, EBSCO*host*, viewed 18 February 2016.

Mazanatti Nunes, William Markito Oliveira, Fabio. Getting Started with Oracle WebLogic Server 12c: Developer's Guide. Birmingham, GBR: Packt Publishing Ltd, 2013. ProQuest ebrary. Web. 18 February 2016.

Schildmeijer, M 2011, *Oracle Weblogic Server 11G PS2 Administration Essentials. [Electronic Book] : Install, Configure, Deploy, And Administer Java EE Applications With Oracle Weblogic Server*, n.p.: Birmingham, UK : Packt Pub., 2011., University of Liverpool Catalogue, EBSCO*host*, viewed 19 February 2016.

Castillo R. (2016) Weblogic Multitenant [Online document] Available from: http://blog.sysco.no/multitenant,/platform/as/a/service/Multitenant/ (Accessed: 19 February 2016)

Castillo R. (2015) JDBC Connection leaks – Generation and Detection [BEA-001153] [Online document] Available from: http://blog.sysco.no/db/locking/jdbc-leak/ (Accessed: 19 February 2016)