# Integration and Configuration of SoftwareAG’s webMethods Broker with JBOSS EAP 6.1

## Table of Contents:

- Install/Configure webMethods Broker Resource Adapter on JBOSS EAP 6 ............... 3
  - RA Deployment ........................................................................................................ 3
  - RA Configuration for Inbound Listeners (MDBs) ...................................................... 3
  - RA Configuration for outbound connections ......................................................... 5
- Extra client configurations through webMethods Broker system properties .......... 10
- Configuring SSL Communication / Authentication .................................................. 13
  - Broker Server SSL Configuration ........................................................................... 13
    - Upload Server TrustStore certificate: ................................................................. 13
    - Load Broker Server Keystore certificate: ......................................................... 14
    - Configure SSL Identity for Broker Server Instance ........................................... 15
    - Connection Groups: Enforcing SSL connections and clients Identities ............ 17
- Clients Configuration (JBOSS) .................................................................................. 18
- Troubleshooting SSL ............................................................................................... 19
- Other Considerations ............................................................................................... 20
- Create a static global module for WebMethods Broker client libraries (Optional) .... 22
  - Module Creation ................................................................................................... 23
  - Module Registration .............................................................................................. 23
- Set MDBs default Resource Adapter and Pool (Optional) ...................................... 25
  - MDBs default Resource Adapter ........................................................................... 25
  - MDBs default Resource Adapter: default pooling ............................................. 25
- Specify Resource Adapters and Pools directly in MDB configurations .................. 26
  - Explicit MDB Resource Adapter binding ............................................................. 26
  - Explicit MDB Pool ................................................................................................ 26
- Reference Documentation ......................................................................................... 27
  - JBOSS EAP 6 .......................................................................................................... 27
Revision History

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Description</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>7/14/2014</td>
<td>1.0</td>
<td>Initial Draft</td>
<td>Fabien Sanglier</td>
</tr>
</tbody>
</table>
Install/Configure webMethods Broker Resource Adapter on JBOSS EAP 6

RA Deployment
- The RAR package is at <WM_BROKER_HOME>/lib/webm-jmsra.rar
- Deployed RAR package onto JBOSS using either way:
  - Copy to the JBOSS deployment folder (<JBOSS-EAP-HOME>/standalone/deployments) and the RAR should be deployed automatically
    - A file “webm-jmsra.rar.deployed” should be created.
    - If nothing is created, or a file “webm-jmsra.rar.failed” is created, an error occurred during deployment.
  - Use the JBOSS admin console to deploy the package just like you would do it for any other deployable resource (EAR, WAR, etc…)
    - Success or failure should be displayed in the console

RA Configuration for Inbound Listeners (MDBs)

Using the JBOSS admin console, we need to register the newly deployed Resource Adapter. Here are steps:

Navigate to the Resource Adapter screen (in “Profile>Connector”)

Create a new Resource Adapter, specifying the same name as the deployed adapter you want to activate. Should be: “webm-jmsra.rar”. Specify either NoTransaction, or XATransaction, as appropriate for your use-case.
When saved, new resource adapter should be shown in the resource adapter grid:

Next, we’ll be adding some properties for the resource adapter. Click on the “properties” tab, and “Add” button:

You can add various properties as required by your deployment and usage, such as:
- JndiProperties (recommended so you don’t have to add it to all your MDBs):
- `java.naming.factory.initial=com.webMethods.jms.naming.WmJmsNamingCtxFactory,java.naming.provider.url=wmjmsnaming://$jndi_broker_name@$jndi_broker_host:$jndi_broker_port`
- **Supports XA:**
  - false (or true if you want to use XA)
- etc...

**Note:**
$jndi_broker_name$: the name of the broker you want to do JNDI lookup on.
$jndi_broker_host$: the network destination of the broker JNDI (IP, DNS name)
$jndi_broker_port$: the listening port of the broker JNDI (e.g. 6849)

### RA Configuration for outbound connections

Back to the Resource Adapter home screen, it’s time to create a connection definition for outbound JMS connection pooling.

Click on “View” link to enter the resource Adapter connection definition configuration screen:

Then, click “add” to create a new outbound Connection definition:
In the connection creation screen:

- **JNDI name**: any based on your naming standards, eg:
  - java:/jms/broker
- **Connection class**:
  - com.sun.genericra.outboundManagedJMSConnectionFactory

As shown in screenshot below:

Once created, the new connection shows in the “Available Connection Definitions” for this resource adapter.
As you can see, the connection is not enabled yet. We will enable it as a final steps after all configurations are specified for that connection.

To configure this connection definition, we will need to add at least 2 mandatory properties:

- JNDIProperties
- ConnectionFactoryJNDIName

Click on the “properties” tab:

- **Add the JNDIProperties parameter** (JNDI connection parameter for webMethods Broker, comma-separated):
  - `java.naming.factory.initial=com.webMethods.jms.naming.WmJmsNamingCtxFactory,java.naming.provider.url=wmjmsnaming://$jndi_broker_name@$jndi_broker_host:$jndi_broker_port`
Note:
$jndi\text{\_broker\_name}$: the name of the broker you want to do JNDI lookup on.
$jndi\text{\_broker\_host}$: the network destination of the broker JNDI (IP, DNS name)
$jndi\text{\_broker\_port}$: the listening port of the broker JNDI (eg. 6849)

- **Add a ConnectionFactoryJNDIName** (Connection Factory in webMethods Broker that should be used by this connection)
  - OutboundQueueConnectionFactory

Connection Properties created should be displayed in the “properties” tab:

Now, we can configure the “pool” for that connection:
Click Edit to specify the right pooling settings for your target throughput and performance:

- min-pool-size
- max-pool-size
- prefetch
- use-strict-min
- flush-strategy

Finally, let’s enable the connection by going back to the “Attributes” tab, clicking “edit” button, and checking “Enabled”:

If everything is successful, the connection should now show as “Enabled” in the connection list and successful trace should show in JBOSS server logs.
Extra client configurations through webMethods Broker system properties

In addition to the RA-specific configuration, webMethods Broker client libraries can be tuned through the use of specific system properties. Please refer to the webMethods Broker API for JMS (JavaDoc) for a complete list of such properties and additional description in the webMethods Broker Messaging Guide.

We will identify here the ones most commonly used:

- `com.webMethods.jms.clientIDSharing (true/false)`
  - Set to true: Very useful to scale out the consumption of messages from a single queue across multiple clients (eg. Multiple JBOSS instances).

If in-order messaging by sending client is not needed, to maximize throughput across instances, set `sharedStateOrdering=none` on the queue administered object.

- Set to false: only 1 client is allowed to bind to any single queue. If a queue is already bound to a client, any other client trying to connect to that same queue will receive an exception.

When `ClientIDSharing` enabled (set to true), you should see in webMethods Broker UI that the single queue ("simplequeue" in our case) has multiple sessions opened to it, one per JVM started.

In our test, we started 2 JBOSS instance and saw 2 sessions connected to that single "simplequeue".
• com.webMethods.jms.keepAliveInterval (integer value in seconds)
  ○ Specifies the time interval in seconds for sending the keep-alive messages from the JMS client to the Broker Server. Sending keep-alive messages in regular intervals helps in identifying dead connections. This feature is very useful when you configure your messaging system in a wide area network. Set the value to 0 (zero) to disable the keep-alive feature. The default value is 30.

• com.webMethods.jms.keepAliveAttempts (integer value)
  ○ Specifies the number of keep-alive attempts. JMS client assumes that the Broker Server connection is down if the Broker Server fails to respond to all the consecutive keep-alive attempts. The default value is 2.

To add these properties to JBOSS, go to the “General Configuration > System Properties” and use the “Add” button.
System Properties

These properties are available throughout the configuration. The -D option specifies if a property should be passed into the JVM start (-DpropertyName=value).

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.webmethods.jms.clientIDSharing</td>
<td>true</td>
<td>Remove</td>
</tr>
<tr>
<td>com.webmethods.jms.keepAliveAttempts</td>
<td>10</td>
<td>Remove</td>
</tr>
<tr>
<td>com.webmethods.jms.keepAliveInterval</td>
<td>15</td>
<td>Remove</td>
</tr>
<tr>
<td>com.webmethods.jms.log.level</td>
<td>ERROR</td>
<td></td>
</tr>
</tbody>
</table>
Configuring SSL Communication / Authentication

Broker Server SSL Configuration

We will be using the webMethods Broker UI (Broker UI component in “My webMethods” portal) to enable SSL communication (add the necessary certificates). Please refer to Chapter 13 – Managing Broker Security of the webMethods Broker Administration Guide for additional information.

Upload Server TrustStore certificate:

Administration > Messaging > Settings > (Tab) SSL Truststore > Add Truststore:

There are 2 choices to load the SSL TrustStore file:
- Upload Truststore from the machine hosting Broker Server.
- Add the Truststore using full path name from the machine hosting My webMethods Server.

Note: The Truststore must be a PEM format.

Either method will work, and once added, you should see the following list with your newly loaded SSL Truststore:
Load Broker Server Keystore certificate:

Administration > Messaging > Settings > (Tab) SSL Keystore > Add Keystore:

There are 2 choices to load the SSL KeyStore file:
- Upload keystore from the machine hosting Broker Server.
- Add the keystore using full path name from the machine hosting My webMethods Server.

Note: The KeyStore can either be a PEM or PKCS12 format.
Either method will work, and once added, you should see the following list with your newly loaded SSL Keystore:

**Configure SSL Identity for Broker Server Instance**

Select the broker server to which to assign an SSL Identify and change the configuration.
Assign the Server Keystore and Server Truststore. Valid types are:

Server Keystore: PKCS12, PEM
Server Truststore: PEM

To configure an SSL identity for a Broker Server, you open its keystore entering the password, select the DN to use as the Broker Server's identity, and assign that DN to the Broker Server.

The SSL tab on the Broker Server Details page for the Broker Server displays information about the SSL configuration. The status should read:

Secure Sockets Layer is configured and working
Connection Groups: Enforcing SSL connections and clients Identities

If you need to enforce SSL encryption on all connections to 1 or more specific queue/topic object(s) (and as such, make all unencrypted connection fail), you must check the “Encryption” box in the “Connection Group” assigned to the queue/topic objects you need to protect (See screenshot below with the “SimpleConnectionGroup” that we used on our test “simplequeue” object).

When that connection group is set with “Encryption”, SSL connection will need to be established properly for any client to connect to the queue/topic protected by this connection group.

If you want to go further, by not only enforcing SSL, but also making sure that only a specific username connects to specific queues/topics, you can also specify the right ACL in that connection group.
Clients Configuration (JBOSS)

To configure the clients (jboss instances) for SSL, we simply need to make the SSL certificates available. This is done through the following webMethods Broker system properties:

- `com.webmethods.jms.ssl.keystore`  
  - Path to the KeyStore file
- `com.webmethods.jms.ssl.keystoretype`  
  - KeyStore type (PKCS12 is the default and only supported)
- `com.webmethods.jms.ssl.truststore`  
  - Path to the TrustStore
- `com.webmethods.jms.ssl.truststoretype`  
  - TrustStore type (JKS is the default and only supported)
- `com.webmethods.jms.username`  
  - JMS SSL authentication username (should be same as Keystore)
- `com.webmethods.jms.password`  
  - JMS SSL authentication password (should be Keystore password)

In order to set these properties in JBOSS using the administration console, go to “General Configuration > System Properties” and add all the properties needed.
JBOSS needs to be restarted to activate these new properties.

Note if the same properties are set on the JNDI Connection Factory they will take precedence over these client side system properties. And the sslEncrypted=true attribute on the connection factory (or specified through My webMethods Server UI) is encrypting the jms SSL connection.

**Troubleshooting SSL**

SSL Encryption and SSL Authentication can be verified in the My webMethods Server (MWS) Admin UI for the Broker Server and the JMS clients.

For additional jms client debugging the following system properties can be set:
- com.webmethods.jms.log.level=DEBUG
- javax.net.debug=ssl

For additional webMethods Broker Server SSL logging (useful to see actual SSL handshake and see errors such as expired certificate or bad certificate chain), add the following can be added to the awbroker.cfg file and broker server restarted:

```
diag=1
eextra-args=-blah +ssl +awssl +result +operation +operationv6 +resultv6
redirect-output=sslaudit.log
```

The “sslaudit.log” file will be written to the broker data directory. Do not provide a full path name for the “redirect-output” property, just the file name.
Other Considerations

It is recommended to define separate queue connection factories for the inbound receiving JMS clients (broker server delivering messages to JMS clients e.g. MDBs) and the outbound sending JMS clients (JMS clients sending messages to broker server) with different application names for better identification within the My webMethods Server UI.

The jmsadmin (command line alternative to My webMethods Server UI) script below provides an example.

```
set $broker="mybroker"
set $server="wmvm:6849"

# create the Broker
connect to broker $broker server $server

# initialize the Broker for JMS
initialize $broker

# bind the administered objects into JNDI
bind queue simplequeue with QueueName=simplequeue sharedState=true sharedStateOrdering=none

bind qcf "InboundQueueConnectionFactory" with brokerHost=$server brokerName=$broker
group=SimpleQueueConnectionGroup useXA=false sslEncrypted=true clientId=simplequeueclient application=mdb-listener

bind qcf "OutboundQueueConnectionFactory" with brokerHost=$server brokerName=$broker
group=SimpleQueueConnectionGroup useXA=false sslEncrypted=true application=msg-pooled-producer

create group SimpleQueueConnectionGroup

# create the topic and client group on the Broker
create queue for simplequeue using InboundQueueConnectionFactory
```

The above jmsadmin script results in the following My webMethods Server screenshots.

![My webMethods Server screenshots](image)

Note, in the screenshots below, SSL Encryption is set to “true” and the Keystore/Truststore information is intentionally empty because they are set via system properties on the JBoss client side.
In the screenshot below, notice the individual application names and client id are displayed.
And lastly, in the client details page, the full SSL jms client connectivity can be verified.

Create a static global module for WebMethods Broker client libraries (Optional)

Due to the way JBOSS AS 7.x (EAP 6.1) loads libraries (module classloader), the WebMethods Broker client libraries currently embedded in the RAR package are not accessible to the EAR/WAR deployments on JBOSS.

To avoid having to embed the WebMethods Broker client libraries in all your “Message Driven Beans” deployments, it is recommended practice to register the m Broker client libraries in a global JBOSS module.
Module Creation

- Navigate to <JBoss-EAP-HOME>/modules/system/layers/base/
- Create directory structure as follow:
  - “com/softwareag/webMethods/broker/main/”
- Navigate to newly create directory
- Copy the necessary webMethods Broker client libraries from your Broker installation direction (<WM_BROKER_HOME>/lib>):
  - wm-brokerclient.jar
  - wm-jmsclient.jar
  - wm-jmsnaming.jar
  - wm-g11nutils.jar
- Create a file named “module.xml” with the following content:

```xml
<module xmlns="urn:jboss:module:1.1" name="com.softwareag.webMethods.broker">
  <properties>
    <property name="jboss.api" value="private"/>
  </properties>

  <resources>
    <resource-root path="."/>
    <resource-root path="wm-brokerclient.jar"/>
    <resource-root path="wm-jmsclient.jar"/>
    <resource-root path="wm-jmsnaming.jar"/>
    <resource-root path="wm-g11nutils.jar"/>
  </resources>

  <dependencies>
    <module name="javax.api"/>
    <module name="javax.jms.api"/>
  </dependencies>
</module>
```

Module Registration

In your JBoss configuration file (standalone.xml, domain.xml), we’ll need to register that newly created module so it’s loaded and available to all clients needing it.

- Stop JBOSS if running
- Edit JBOSS configuration file (standalone.xml, domain.xml)
- Navigate to subsystem section “urn:jboss:domain:ee:1.1”
- If not there yet, create a “global-modules” section
- In that “global-modules” section, add the newly created WebMethods Broker module identification: <module name="com.softwareag.webMethods.broker" slot="main"/>
• Restart JBOSS

The WebMethods Broker client libraries should now be loaded in JBOSS’s classpath for easy usage in all your deployments.
Set MDBs default Resource Adapter and Pool (Optional)

To ensure that all deployed MDBs use the same Resource Adapter and pooling configurations, it’s useful to set them up as defaults in JBOSS configuration.

**MDBs default Resource Adapter**

In your JBoss configuration file (standalone.xml, domain.xml):

- Stop JBOSS if running
- Edit JBOSS configuration file (standalone.xml, domain.xml)
- Navigate to subsystem section “urn:jboss:domain:ejb3:1.4”
- In that section, find the “<mdb></mdb>” section.
- Edit the “resource-adapter-ref” line and set the attribute “resource-adapter-name” to the WebMethods Broker resource adapter registered in the previous section (“webm-jmsra.rar”)
- The final result should be as follow:

```xml
<subsystem xmlns="urn:jboss:domain:ejb3:1.4">
  ...
  <mdb>
    <resource-adapter-ref resource-adapter-name="webm-jmsra.rar"/>
    <bean-instance-pool-ref pool-name="mdb-strict-max-pool"/>
  </mdb>
  ...
</subsystem>
```

- Restart JBOSS

Now, any MDBs deployed to JBOSS will use the WebMethods Broker resource adapter by default.

**MDBs default Resource Adapter: default pooling**

To achieve target throughput / performance, it’s critically important to properly tune the amount of MDB instances that should be actively consuming messages. Such tuning is done through MDB pooling.

- In that same “<mdb></mdb>” section, you can see that the “bean-instance-pool-ref” tag reference a pool named “mdb-strict-max-pool”
- Refer to the “<pools> </pools>” section to find the detailed configuration for that pool “mdb-strict-max-pool”
- Modify it accordingly. (Attribute “max-pool-size” is particularly important to get right to achieve best performance)
Specify Resource Adapters and Pools directly in MDB configurations

Explicit MDB Resource Adapter binding
If multiple Resource Adapters are deployed and used with various MDB deployments, it’s not possible to use a single default resource adapter. If that’s the case, you might want to explicitly define which MDB use which Resource Adapter.

That is done through your MDB deployment configuration (within your deployed application).
Using annotations, simply add the JBOSS-specific @ResourceAdapter annotation (org.jboss.ejb3.annotation.ResourceAdapter) to your MDB(s) to explicitly use a specific resource adapter.

For example, to bind to the configured WebMethods Broker resource adapter:
@ResourceAdapter("webm-jmsra.rar")

Explicit MDB Pool

Similarly, it might not be practical to use the same default pool for all your MDBs. Again, you can specify which JBOSS-defined pool to use directly in your MDB code using the following @Pool annotation (org.jboss.ejb3.annotation.Pool)

For example, to have your MDB use a different pool (eg “mdb-small-strict-max-pool”) from the default:
@Pool(value="mdb-small-strict-max-pool")
Reference Documentation

JBoss EAP 6
Administration and Configuration guide

Installation Guide

Development Guide

Security Guide

API documentation

webMethods Broker

webMethods Broker Documentation
http://documentation.softwareag.com/webmethods/broker/pif823/index.htm
http://documentation.softwareag.com/webmethods/broker/pif96/index.htm

webMethods Broker Administration Guide
http://documentation.softwareag.com/webmethods/wmsuites/wmsuite8-2_sp2/Broker/8-2-SP3_Administering_Broker.pdf

webMethods Broker Messaging Programmer's Guide

webMethods Broker API for JMS (JavaDoc)