Welcome to...

Apache MQ to JBoss Integration

featuring JBoss 6 EAP / 7.x.x and Apache Active MQ 5.6/7

Cav Edwards

Swarm Software – Research Laboratories

Table of Contents

1 Overview

This document explains how to set-up and configure Apache MQ for seamless integration with JBoss, as an embedded messaging broker. The high-availability section considers connecting JBoss to grids of standalone ActiveMQ nodes.

1.1 JBoss

To enable Apache MQ in JBoss you are required to edit an xml file within <JBoss installation>\standalone\configuration. These XML files contain the configuration for the JBoss server.
The nightly builds are available here for version 7.
https://hudson.jboss.org/jenkins/view/JBoss%20AS/job/JBoss-AS-7.0.x/

Please download the jboss-7.0.x.zip.
These instructions apply to 6.0.0 EAP too.

1.2 Apache MQ Standalone Server

Apache provides a standalone Active MQ solution. This is required if you would also like to use a database of your choice, or would like to implement High Availability ActiveMQ Grids / Clusters.
Download the standalone here :
https://activemq.apache.org/download.html

1.3 Apache MQ Resource-adapter

The Active MQ Resource-adapter provides the embedded implementation of Active MQ as a Resource Archive, RAR file. At the time of writing, it is easier to use version 5.6. 5.7 snapshots are available and these should work too, but 5.6 appears to be the latest supported version.
Download the Resource-adapter RAR file here :
http://repo1.maven.org/maven2/org/apache/activemq/activemq-rar/

Drop the Resource Adapter RAR file in the deployment folder for JBoss:
<JBoss installation>\standalone\deployments

This file will be edited later.

2 Configuration

Consider the following overview.

JBoss has a configuration file standalone.xml which can be used to integrate Active MQ. Other configuration files exist for clustered configuration for example.

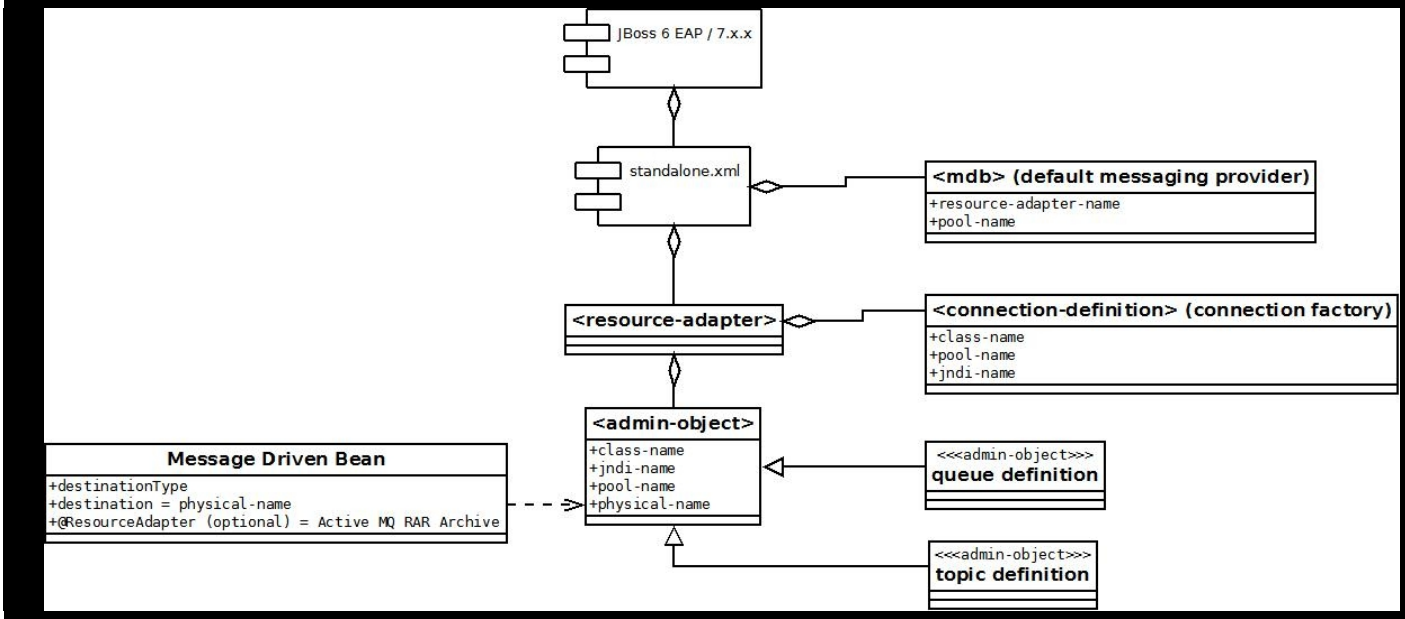
The JBoss configuration file needs to contain a resource-adapter configuration. Its within this resource-adapter that Apache Active MQ is integrated.

An MDB specification in standalone.xml can make Apache Active MQ the default messaging provider.

The resource adapter configuration needs to contain a connection factory, used to obtain connections to Queues and Topics.

For each Queue or Topic there needs to be at least an implementation class name, a JNDI name, a physical name and a pool name.

The MDB is attached to the queue via its destination being set to the physical-name for the topic or queue admin-object.



2.1 JBoss Resource Adapter Configuration

The configuration above corresponds to the XML below, showing a single queue and single topic defined as part of the resource adapter configuration. These resources are referred to as Admin Objects by JCA.

```
<subsystem xmlns="urn:jboss:domain:resource-adapters:1.0">
  <resource-adapters>
    <resource-adapter>
        <archive>
            activemq-rar-5.6.rar
        </archive>
        <transaction-support>XATransaction</transaction-support>
        <config-property name="UseInboundSession">
            false
        </config-property>
        <config-property name="Password">
            password
        </config-property>
        <config-property name="UserName">
            admin
        </config-property>
        <config-property name="ServerUrl">
            vm://localhost?brokerConfig=xbean:broker-config.xml
        </config-property>
        <connection-definitions>
            <connection-definition class-name="org.apache.activemq.ra.ActiveMQManagedConnectionFactory" jndi-
name="java:jboss/exported/ConnectionFactory" enabled="true" use-java-context="true" pool-name="ConnectionFactory"/>
        </connection-definitions>
        <admin-objects>
            <admin-object class-name="org.apache.activemq.command.ActiveMQQueue" jndi-name="java:/queue/FromETSQueue"
use-java-context="true" pool-name="queue1">
                <config-property name="PhysicalName">
                    queue1
                </config-property>
            </admin-object>
            <admin-object class-name="org.apache.activemq.command.ActiveMQTopic" jndi-
name="java:jboss/eis/ao/ActiveMQTopic" use-java-context="true" pool-name="topic1">
                <config-property name="PhysicalName">
                    topic1
                </config-property>
            </admin-object>
        </admin-objects>
    </resource-adapter>
  </resource-adapters>
</subsystem>
```

2.1.1 Embedded Broker

The setting:

```
vm://localhost
```

will connect to the default embedded broker. If you elect to customise the broker configuration then replace vm://localhost with vm://<your broker name>, using the broker name from broker-config.xml. You will need to edit broker-config.xml if you choose to change the default persistence mechanism or to integrate with a standalone Apache Active MQ.
Active MQ will run in embedded mode with vm://localhost.

2.1.2 Standalone Broker

For a broker running on brokerHostName at port 61616, the stanza would look as follows:

```
tcp://brokerHostName:61616
```

For a failover specification to multiple brokers operating in a cluster, consider the following example stanza.

```
failover:(tcp://broker1:61616,tcp://broker2:61616,tcp://broker3,tcp://broker4:61616)?initialReconnectDelay=100
```

2.2 Defaulting the messaging provider

An MDB specification is needed to instruct the JBoss container to use a particular Resource Adapter Messaging Provider as the default. If you do not specify this then you will need to instruct the MDB which Messaging system you require, if it is not to use the default messaging provider. See the MDB Registration with Apache Active MQ section.

```
    <subsystem xmlns="urn:jboss:domain:ejb3:1.3">
    <datasources>
            <mdb>
                    <resource-adapter-ref resource-adapter-name="activemq-rar-5.6.rar" />
                    <bean-instance-pool-ref pool-name="mdb-strict-max-pool" />
            </mdb>
    </datasources>
</subsystem>
```

3 JCA registers the admin objects

3.1 Registration of the Connection Factory by JCA

Registration of the Connection Factory by JCA looks as follows.

```
12:33:45,420 INFO  [org.jboss.as.deployment.connector] (MSC service thread 1-8) JBAS010406: Registered connection
factor
y java:jboss/exported/ConnectionFactory
```

3.2 Registration of the Queue / Topic by JCA

Registration of the Topics and Queues by JCA looks as follows.

```
12:33:45,423 INFO  [org.jboss.as.deployment.connector] (MSC service thread 1-8) JBAS010405: Registered admin object at
java:jboss/queue/fromSomeWhereQueue
```

4 MDB Registration with Apache ActiveMQ

The MDB is typically attached to the queue within an EAR project.
The Message Driver Bean (MDB) needs to be bound to the physical name provided from the standalone.xml JBoss Server configuration file.

```
@MessageDriven(
        activationConfig = {
                @ActivationConfigProperty(propertyName="destinationType",     propertyValue="javax.jms.Queue"),
                @ActivationConfigProperty(propertyName="destination",         propertyValue="fromSomewhereQueue"),
                @ActivationConfigProperty(propertyName="acknowledgeMode",   propertyValue="Auto-acknowledge"),
        })
@ResourceAdapter("activemq-rar-5.6.rar")
```

Note: The @ResourceAdapter annotation is only required when the default messaging provider is not Apache Active MQ.
Note: The destination within the @MessageDriven annotation must be the physical name taken from standalone.xml and not the JNDI name as it is in Hornet Queue configuration.

5 Example JMS Client Code

Most will be familiar with this code. Here we launch a JMS message onto a queue.

```
oConnectionFactory = (QueueConnectionFactory)oInitialContext.lookup("java:jboss/exported/ConnectionFactory");
Connection oConnection = oConnectionFactory.createConnection();
```

```
Session oSession = oConnection.createSession(false, Session.AUTO_ACKNOWLEDGE);
Destination oDestination = (Queue) oInitialContext.lookup("java:jboss/queue/fromSomeWhereQueue");
MessageProducer oMessageProducer = oSession.createProducer((Queue)oDestination);
TextMessage oTextMessage = oSession.createTextMessage();
oTextMessage.setText("Hello Apache MQ Guru's");
oMessageProducer.send(oTextMessage);
```

6 Output when the MDB OnMessage() method fires

On a successful message reception the message captured by the MDB will look as follows.

```
16:27:36,655 INFO  [stdout] (default-threads - 3) ActiveMQTextMessage {commandId = 13, responseRequired = true,
messageId = ID:SWEETCORN-52564-1343230036458-5:1:2:1:1, originalDestination = null, originalTransactionId = null,
producerId = ID:SWEETCORN-52564-1343230036458-5:1:2:1, destination = queue://FromETSQueue, transactionId = null,
expiration = 0, timestamp = 1343230056646, arrival = 0, brokerInTime = 1343230056649, brokerOutTime = 1343230056651,
correlationId = null, replyTo = null, persistent = true, type = null, priority = 4, groupID = null, groupSequence = 0,
targetConsumerId = null, compressed = false, userID = null, content = org.apache.activemq.util.ByteSequence@163a3c40,
marshalledProperties = null, dataStructure = null, redeliveryCounter = 0, size = 1052, properties = null,
readOnlyProperties = true, readOnlyBody = true, droppable = false, text = Hello Apache MQ Guru's}
```

7 Database Persistence

Apache MQ can persist data to a database of your choice. It can also journal and can use the journal for high-performance check-pointing. Your use of these facilities vary according to your transactional and performance requirements. To support XA and to cater for system failure you ought to architect a solution with Database Persistence to a database that is still available in the event your JBoss server has stopped.

7.1 DB Lock Timeout

You may need to add the following stanza if you are using db persistence. This prevents active MQ timing out its DB lock when using DB persistence.

```
<amq:databaseLocker>
  <amq:database-locker queryTimeout="-1" />
</amq:databaseLocker>
```

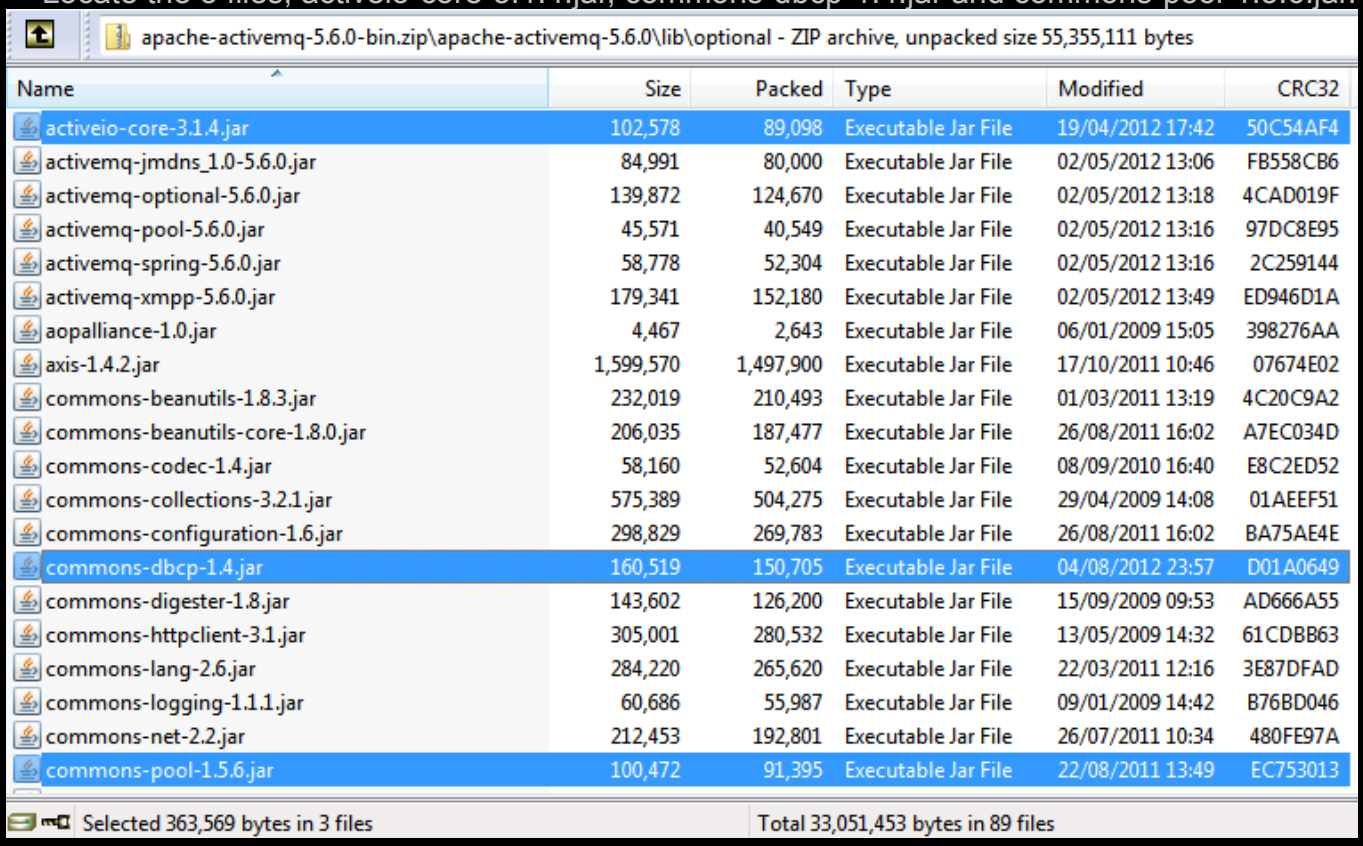
7.2 Modify the Resource Adapter RAR file

The RAR file needs to be modified for use with JBoss.

1. Copy broker-config.xml from the RAR archive. Place the copy in the <jboss install directory> folder.
2. Delete the broker-config.xml file from the root of the archive. The solution will use the broker-config.xml configured in the JBoss folder.

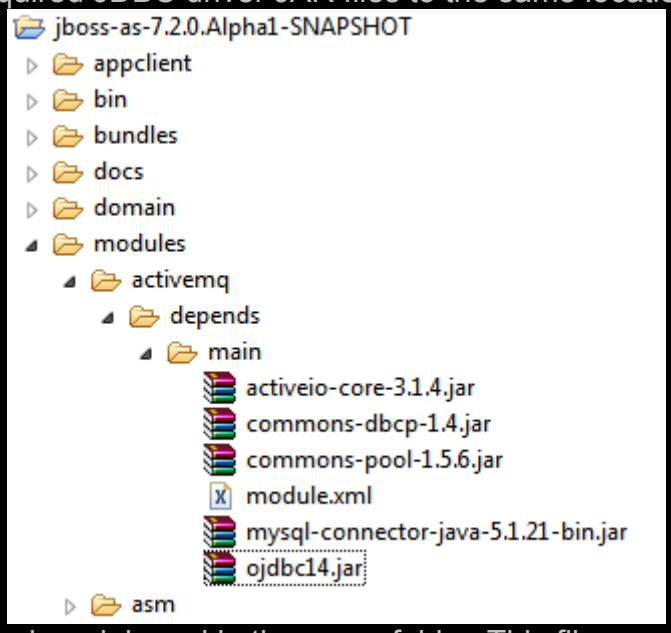
7.3 Create a JBoss module

1. Using the standalone Active MQ download, within apache-activemq-5.6.0\lib\optional. Locate the 3 files, activeio-core-3.1.4.jar, commons-dbcp-1.4.jar and commons-pool-1.5.6.jar.



2. Create a folder called <jboss install dir>\modules\activemq\depends\main. Copy the 3 files from the archive into the new folder. If you forget to do this step, you will see classloader errors when they are attempted to be found and loaded.

3. Copy all your required JDBC driver JAR files to the same location.

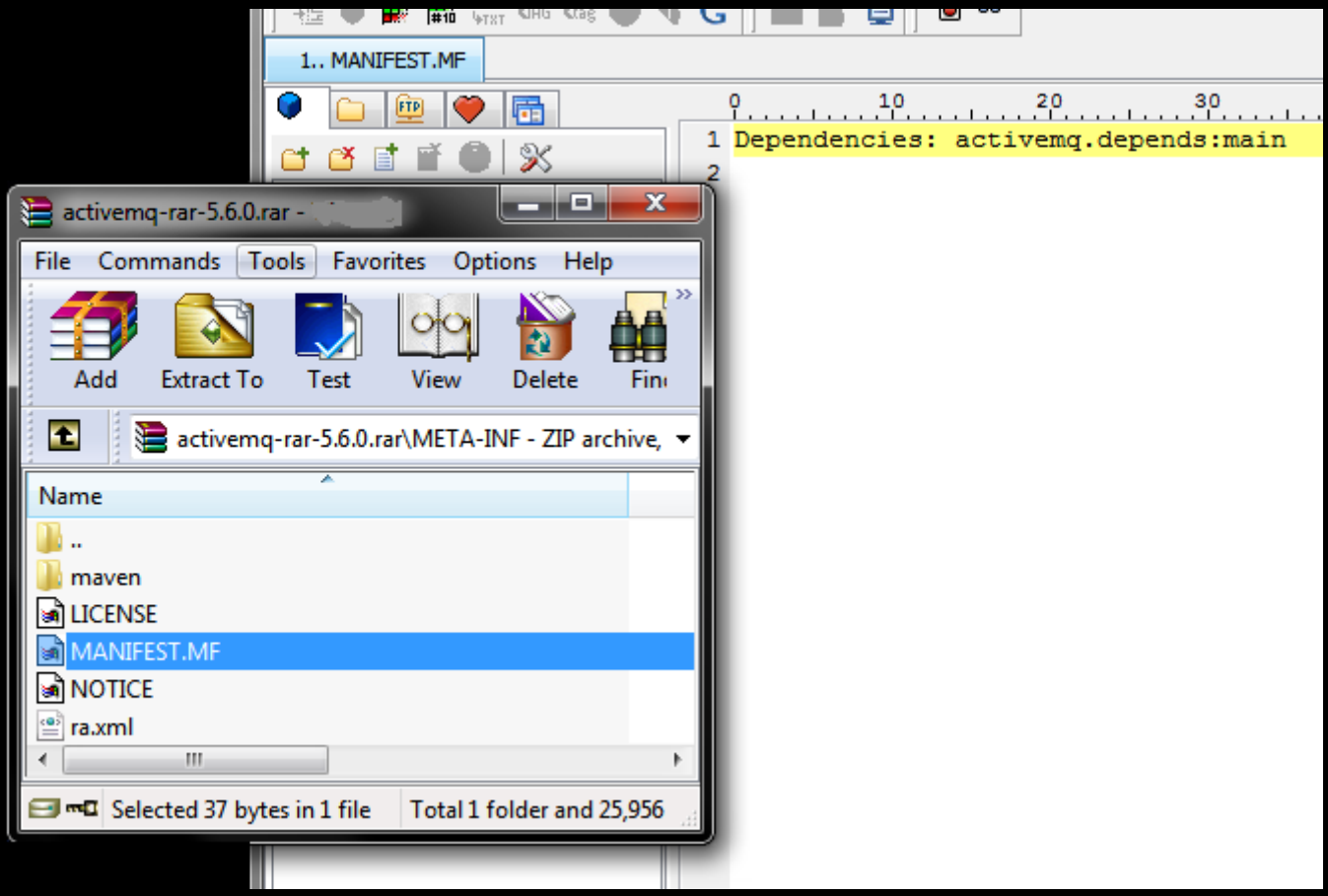


4. Create a file called module.xml in the same folder. This file needs to reflect the JAR files that Active MQ requires, to load successfully, and the JDBC drivers you intend to use.

```
<?xml version="1.0" encoding="UTF-8"?>

<module xmlns="urn:jboss:module:1.1" name="activemq.depends">

   <resources>
           <!-- required Active MQ JAR's -->
           <resource-root path="commons-dbcp-1.4.jar" />
           <resource-root path="commons-pool-1.5.6.jar" />
           <resource-root path="activeio-core-3.1.4.jar" />

           <!-- database drivers -->
           <resource-root path="mysql-connector-java-5.1.21-bin.jar" />
           <resource-root path="ojdbc14.jar" />
           <!-- Insert resources here -->
   </resources>

   <dependencies>
        <module name="javax.api" export="true"/>
   </dependencies>

</module>
```

5. Finally the manifest within the Resource Archive needs to be modified to ensure the dependency on the new module is identified to the class-loader. Open the Resource-adapter Archive. Locate the meta-inf folder\manifest.mf. Add the dependency on the activemq module.

```
Dependencies: activemq.depends:main
```



7.4 Persistence Type

Within the broker-config.xml file datasources and a broker configuration, containing a management context and persistence adapter need to be configured. Transport definitions are ignored so can be removed from broker-config.xml. The transport is decided by the JCA layer in JBoss. See : . In the example below the MySQL datasource is selected for persistence. You can have as many database solutions registered as you like. You can have many bean definitions. Databases in use are specified in the PersistenceAdapter stanza.
NOTE: In the example below the KAHA-DB persistence mechanism is switched off. The MySQL DataSource with journalling is selected for use, within the amq:persistenceFactory.

```
            <!-- Database Persistence with journelling -->
            <amq:persistenceFactory>
                    <amq:journalPersistenceAdapterFactory
                            journalLogFiles="5"  dataDirectory="activemq-data/jounal"
                            dataSource="#mysql-ds" />
            </amq:persistenceFactory>
```

NOTE: The active mq core schema has an amq namespace.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Licensed to the Apache Software Foundation (ASF) under one or more contributor
     license agreements. See the NOTICE file distributed with this work for additional
     information regarding copyright ownership. The ASF licenses this file to
     You under the Apache License, Version 2.0 (the "License"); you may not use
     this file except in compliance with the License. You may obtain a copy of
     the License at http://www.apache.org/licenses/LICENSE-2.0 Unless required
     by applicable law or agreed to in writing, software distributed under the
     License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS
     OF ANY KIND, either express or implied. See the License for the specific
     language governing permissions and limitations under the License. -->
```

```
<!-- START SNIPPET: xbean -->
<beans xmlns="http://www.springframework.org/schema/beans"
     xmlns:amq="http://activemq.apache.org/schema/core" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
  http://activemq.apache.org/schema/core http://activemq.apache.org/schema/core/activemq-core.xsd">

     <!-- Oracle DataSource Sample Setup -->
     <bean id="oracle-ds" class="org.apache.commons.dbcp.BasicDataSource"
             destroy-method="close">
             <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver" />
             <property name="url" value="jdbc:oracle:thin:@192.168.150.107:1521:EFP" />
             <property name="username" value="###username###" />
             <property name="password" value="###password###" />
             <property name="maxActive" value="200" />
             <property name="poolPreparedStatements" value="true" />
     </bean>

     <!-- MySql DataSource Sample Setup -->
     <bean id="mysql-ds" class="org.apache.commons.dbcp.BasicDataSource"
             destroy-method="close">
             <property name="driverClassName" value="com.mysql.jdbc.Driver" />
             <property name="url" value="jdbc:mysql://localhost:3306/activemq?relaxAutoCommit=true" />
             <property name="username" value="###username###" />
             <property name="password" value="###password###" />
             <property name="maxActive" value="200" />
             <property name="poolPreparedStatements" value="true" />
     </bean>

     <!-- shutdown hook is disabled as RAR classloader may be gone at shutdown -->
     <amq:broker persistent="true" brokerName="myAMQBroker"
             useJmx="false" useShutdownHook="false">

             <amq:managementContext>
                     <!-- use appserver provided context instead of creating one, for jboss
                             use: -Djboss.platform.mbeanserver -->
                     <amq:managementContext createConnector="false">
                     </amq:managementContext>
             </amq:managementContext>

             <!-- Database persistence only HA
             If Clustered DB ? Then DR-->
             <!-- <amq:persistenceAdapter> <amq:jdbcPersistenceAdapter createTablesOnStartup="true"
                     dataSource="#mysql-ds"> </amq:jdbcPersistenceAdapter> </amq:persistenceAdapter> -->

             <!-- Journalling HA
             If SAN ? Then DR -->
             <!-- <amq:persistenceFactory>
                     <amq:journalPersistenceAdapterFactory
                             journalLogFiles="5" dataDirectory="activemq-data/journal" />
             </amq:persistenceFactory>-->

             <!-- Database Persistence with journalling HA -
             If SAN AND Clustered DB ? Then DR -->
             <amq:persistenceFactory>
                     <amq:journalPersistenceAdapterFactory
                             journalLogFiles="5"  dataDirectory="activemq-data/journal"
                             dataSource="#mysql-ds" />
             </amq:persistenceFactory>

             <!-- Kaha DB HA - high performance file-system database
             If SAN ? Then DR-->
             <!-- <amq:persistenceAdapter> <amq:kahaDB directory="activemq-data/kahadb"
                     /> </amq:persistenceAdapter> -->
     </amq:broker>
</beans>
```

8 ActiveMQ Clustering / Grids

In this section we consider failover for JMS messaging to increase the reliability of Active MQ in the event of catastrophic failure of a processing node or an entire datacentre.
In the following architecture, use is made of Active MQ Grids and Master / Slave DB persistence. Brokers use Broker Discovery to find each other.

Failover : https://activemq.apache.org/failover-transport-reference.html

Master / Slave DB persistence : https://activemq.apache.org/jdbc-master-slave.html

Broker Discovery : https://activemq.apache.org/discovery-transport-reference.html

NOTE: A consequence of this architecture is the need for each message that is sent by a producer to be retried with a time-out to each JBoss node in the event one JBoss node stops.

8.1 Failover to multiple Brokers

In the high-availability architecture each JBoss node is capable of failing-over to 1 or more brokers. Each node running JBoss also has a broker.
The configuration of the broker within JBoss changes...

from:
```
vm://localhost?brokerConfig=xbean:broker-config.xml
```

to:
```
failover:(tcp://localhost:61616,tcp://remotehost1:61616,tcp://remotehost2:61616)?initialReconnectDelay=100
```

8.2 Broker Discovery

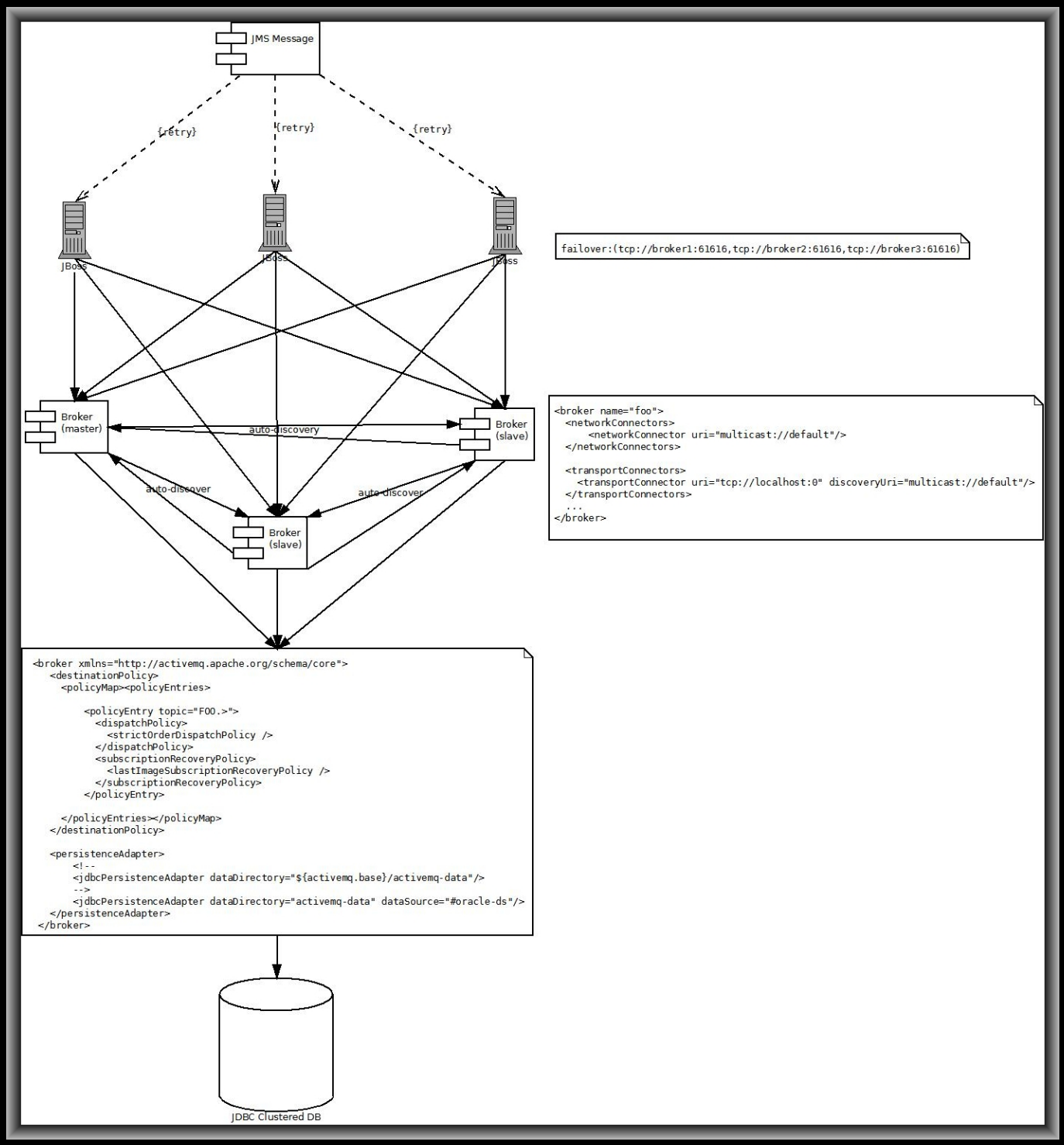
Broker Discovery enables brokers to establish a grid automatically via multi-cast broadcasting. A network connector and transport specification are required to enable this.
In the event of failure, messages are retried to other brokers. The broker name becomes more significant in the cluster as this is the means by which you identify a particular member of the grid / cluster when you are using the Active MQ Console.

```
<broker name="foo">
    <networkConnector uri="multicast://default"
            dynamicOnly="true"
            networkTTL="3"
            prefetchSize="1"
            decreaseNetworkConsumerPriority="true" />
    </networkConnectors>

    <transportConnectors>
            <transportConnector uri="tcp://localhost:0" discoveryUri="multicast://default"/>
    </transportConnectors>
  ...
</broker>
```

When the grid / cluster of brokers initialise they automatically find each other and establish a network connection.

8.3 Broker Master / Slave to the Persistence Mechanism

With Master / Slave persistence, each broker tries to obtain a lock on the database. Once obtained the broker becomes the master and persists messages to a database.
In the event of failure of the master, each remaining slave attempts to get the newly-released lock on the database to become the master. Messages persistence is then resumed.
NOTE: A SAN could be used in place of DB persistence, with a file-system database such as Kaha DB.

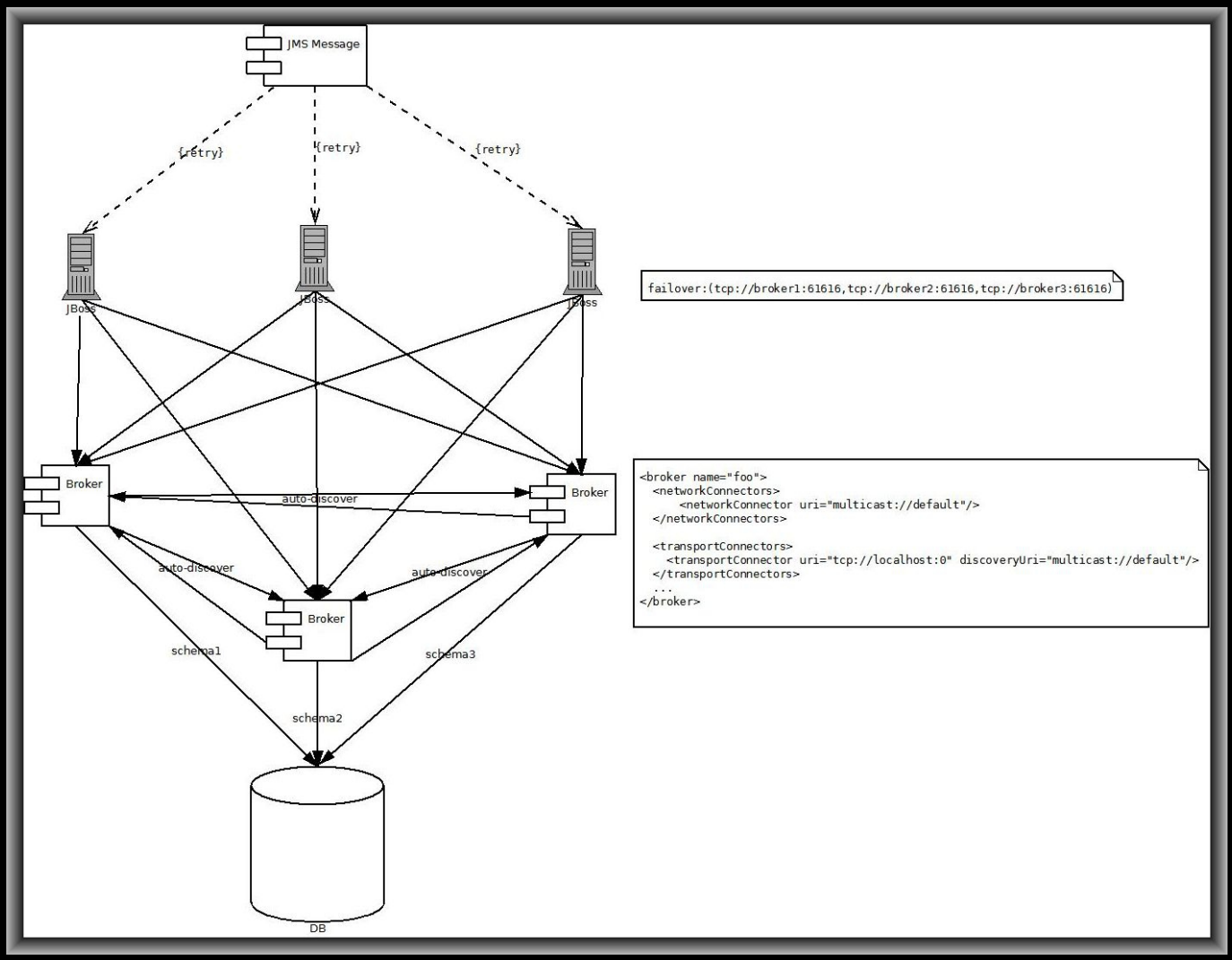


```
<beans>

  <!-- Allows us to use system properties as variables in this configuration file -->
  <bean class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer"/>

  <!--  This xbean configuration file supports all the standard spring xml configuration options -->

  <!-- Postgres DataSource Sample Setup -->
  <!--
  <bean id="postgres-ds" class="org.postgresql.ds.PGPoolingDataSource">
    <property name="serverName" value="localhost"/>
    <property name="databaseName" value="activemq"/>
    <property name="portNumber" value="0"/>
    <property name="user" value="activemq"/>
    <property name="password" value="activemq"/>
    <property name="dataSourceName" value="postgres"/>
    <property name="initialConnections" value="1"/>
    <property name="maxConnections" value="10"/>
  </bean>
  -->

  <!-- MySql DataSource Sample Setup -->
  <!--
  <bean id="mysql-ds" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
    <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
    <property name="url" value="jdbc:mysql://localhost/activemq?relaxAutoCommit=true"/>
    <property name="username" value="activemq"/>
    <property name="password" value="activemq"/>
    <property name="poolPreparedStatements" value="true"/>
  </bean>
  -->

  <!-- Oracle DataSource Sample Setup -->
  <!--
  <bean id="oracle-ds" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
    <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
    <property name="url" value="jdbc:oracle:thin:@localhost:1521:AMQDB"/>
    <property name="username" value="scott"/>
    <property name="password" value="tiger"/>
    <property name="poolPreparedStatements" value="true"/>
  </bean>
  -->

  <!-- Embedded Derby DataSource Sample Setup -->
  <!--
  <bean id="derby-ds" class="org.apache.derby.jdbc.EmbeddedDataSource">
    <property name="databaseName" value="derbydb"/>
    <property name="createDatabase" value="create"/>
  </bean>
  -->

  <broker xmlns="http://activemq.apache.org/schema/core">

    <destinationPolicy>
      <policyMap><policyEntries>

          <policyEntry topic="FOO.>">
            <dispatchPolicy>
              <strictOrderDispatchPolicy />
            </dispatchPolicy>
            <subscriptionRecoveryPolicy>
              <lastImageSubscriptionRecoveryPolicy />
            </subscriptionRecoveryPolicy>
          </policyEntry>

      </policyEntries></policyMap>
    </destinationPolicy>

    <persistenceAdapter>
        <jdbcPersistenceAdapter dataDirectory="${activemq.base}/activemq-data"/>

        <!--
        <jdbcPersistenceAdapter dataDirectory="activemq-data" dataSource="#oracle-ds"/>
        -->
    </persistenceAdapter>
     <networkConnectors>
            <networkConnector uri="multicast://default" />
```

```
     </networkConnectors>

    <transportConnectors>
       <transportConnector name="default" uri="tcp://localhost:61616"/>
    </transportConnectors>
  </broker>
</beans>
```

8.4 Active MQ Grid

Considering the architecture below. If each broker is set to use a separate tablespace, then they will behave as an ActiveMQ grid, with messages load balanced across the grid and fail-over protecting against node failure.



8.5 Message Groups

Message groups are used to ensure only-once ordered processing of messages, with automatic load-balancing and fail-over. The message producer needs to set a message group ID on the message.
Active MQ Message groups are detailed here:
https://activemq.apache.org/message-groups.html

```
Message message = session.createTextMessage("<foo>hey</foo>");
message.setStringProperty("JMSXGroupID", "IBM_NASDAQ_20/4/05");
message.setIntProperty("JMSXGroupSeq", -1);
...
producer.send(message);
```