



**89 Fifth Avenue, 7th Floor**

**New York, NY 10003**

**[www.TheEdison.com](http://www.TheEdison.com)**

**212.367.7400**

**[Info@TheEdison.com](mailto:Info@TheEdison.com)**

## White Paper

---

**IBM WebSphere MQ 7.5 versus  
Apache ActiveMQ 5.9: Failover,  
Transactional Integrity and  
Administration**

Printed in the United States of America

Copyright 2014 Edison Group, Inc. New York.

This report was developed by Edison Group, Inc. with IBM assistance and funding. Edison Group is a custom consultancy, with a long-standing relationship with IBM, prepared to answer specific challenges in sales, marketing, and development. As a technology research and business consulting firm, Edison Group provides competitive analysis, product and solution analysis, product testing, business value and TCO studies, customer and partner research, customer case studies, partner program support, white papers, sales enablement tools, market analysis, and related marketing content.

This report may utilize information, including publicly available data, provided by various companies and sources, including IBM. The opinions are those of Edison Group, Inc. and do not necessarily represent IBM's position.

Edison Group offers no warranty either expressed or implied on the information contained herein and shall be held harmless for errors resulting from its use.

All trademarks are copyright their respective owners.

First Publication: May 2014

Produced by: Bill Karounos, Messaging Specialist; Hanny Hindi, Analyst; Manny Frishberg, Editor; Barry Cohen, Editor-in-Chief

## Table of Contents

---

Executive Summary .....	1
Reliability and Failover .....	2
Transactional Integrity and Management .....	7
Performance .....	11
Installation and Configuration .....	13
Management and Administration .....	14
Appendix 1: Feature Comparison .....	27
Appendix 2: Items Not Covered In This Paper .....	30

## Executive Summary

---

This paper provides a comparison of IBM WebSphere MQ 7.5 and the Apache Software Foundation's ActiveMQ 5.9, with detailed analyses of technical factors including stability, reliability, ease of use, performance, and operational capabilities.

ActiveMQ and WebSphere MQ both meet very basic messaging requirements. However, customers in enterprise environments that need high availability and robust failover should seriously consider WebSphere MQ for the following reasons:

- **Failover:** ActiveMQ lost or duplicated messages during “power outage” and “network failure” scenarios. This is unacceptable in enterprise environments.
- **Documentation:** IBM’s documentation was far more complete and up-to-date than Apache’s, especially with respect to configuration, management, API documentation, and advanced configurations such as clustering, load balancing and high availability.
- **Performance:** In persistent tests, WebSphere MQ performed 60 to 90 percent faster with messages ranging from 256 bytes to 1MB. (Because of network limitations, non-persistent tests were inconclusive, but initial results demonstrated an advantage for WebSphere MQ as well.)
- **Transaction Management:** A major distinction between the two systems was the ease of managing transactions: whereas native WebSphere MQ capabilities allowed us to manage transaction between the database and the messaging server. ActiveMQ requires an external application server with XA support to control 2PC transactions.
- **Administration:** ActiveMQ’s web console provides very limited functionality. For many basic and most of the advanced functions, such as editing queues or changing maximum message size users have to manually edit configuration files. Moreover, ActiveMQ requires a unique URL and separate browser window for each broker, while the WebSphere MQ Explorer allows users to administer multiple brokers from a single interface.
- **Platform Compatibility:** WebSphere MQ is not only “supported,” but fully certified on a wide variety of platforms, from Windows and Linux to Solaris and HP-UX. While ActiveMQ works on many of these platforms, it is *not* specifically certified with these platforms, including System z mainframe, still crucial in many production environments.

If high availability, reliability, usability, thorough documentation, and platform compatibility are NOT important, ActiveMQ may be a good platform. But for enterprise customers with reliability needs, WebSphere MQ is the superior choice.

## Reliability and Failover

---

Edison tested the failover capabilities and reliability of WebSphere MQ and ActiveMQ in two common scenarios: a “power outage” and a “network disruption.”

The “power outage” was simulated by bringing down the virtual machine on which the main server was running. The “network disruption” was simulated by bringing down the network interface controller (NIC). As expected, for the period of the outage some messages were not being delivered from the client machine to the server. The difference came in how the two systems recovered from the disruption.

WebSphere MQ’s recovery was clearly superior:

Testing Scenario	ActiveMQ	WebSphere MQ
<b>“Power Outage”</b>	On average Edison saw about 2 percent duplicate messages, and no lost messages	No lost or duplicate messages
<b>“Network Disruption”</b>	On average Edison saw about 2 percent duplicate messages, and 100 percent-lost messages on restart. ActiveMQ failed to restore the original master-slave relationship among servers after the network was recovered.	No lost or duplicate messages

A key requirement of messaging is reliable delivery of messages *once and only once*. By duplicating and *losing* messages, ActiveMQ violated this key principle.

### ***ActiveMQ Configuration #1: Master-Slave***

- Shared File System (NFSv4)
- Two ActiveMQ brokers running on two VMs in a Master-Slave configuration
  - One server running as the Master broker on VM1
  - One server running as the Slave broker on VM2

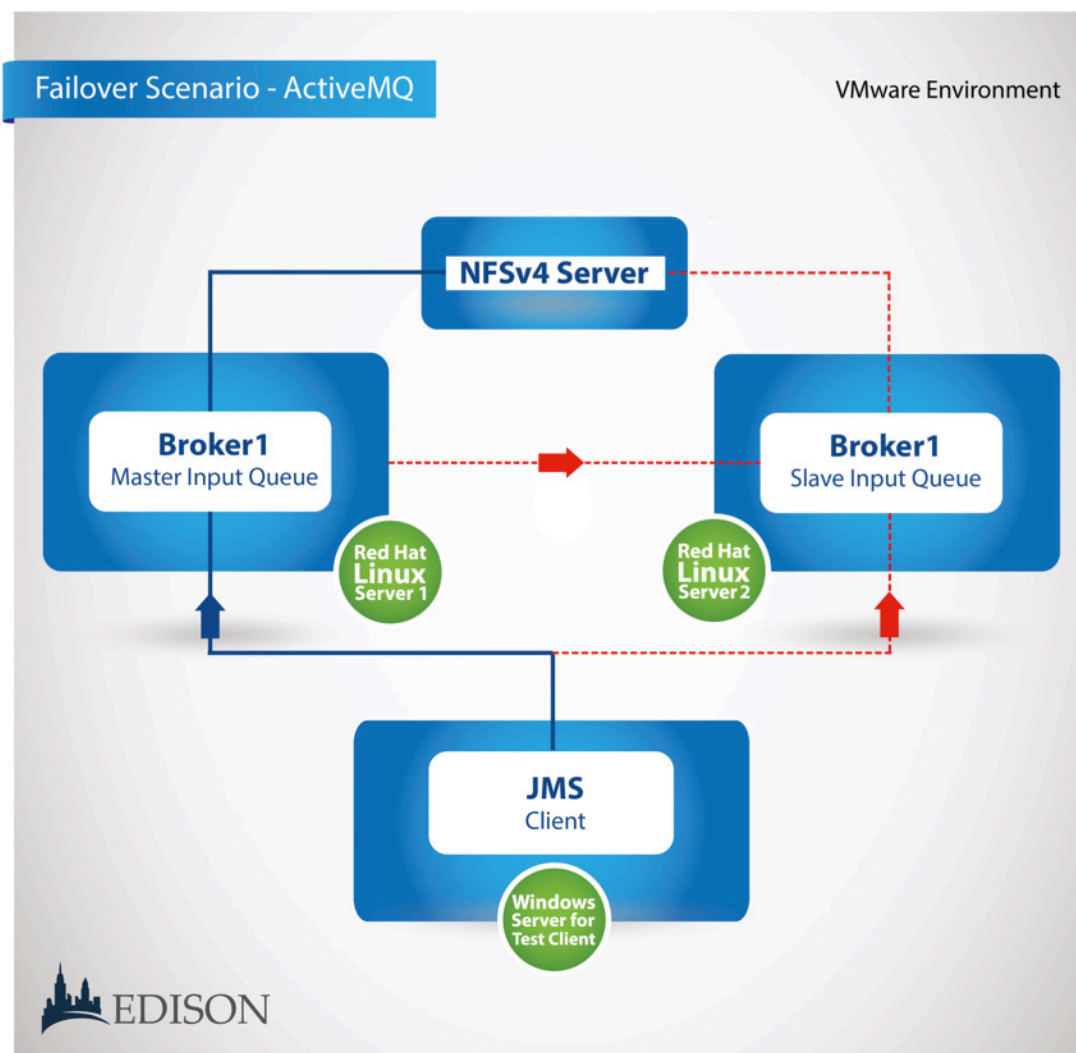


Figure 1: ActiveMQ 5.9 Test Failover Environment

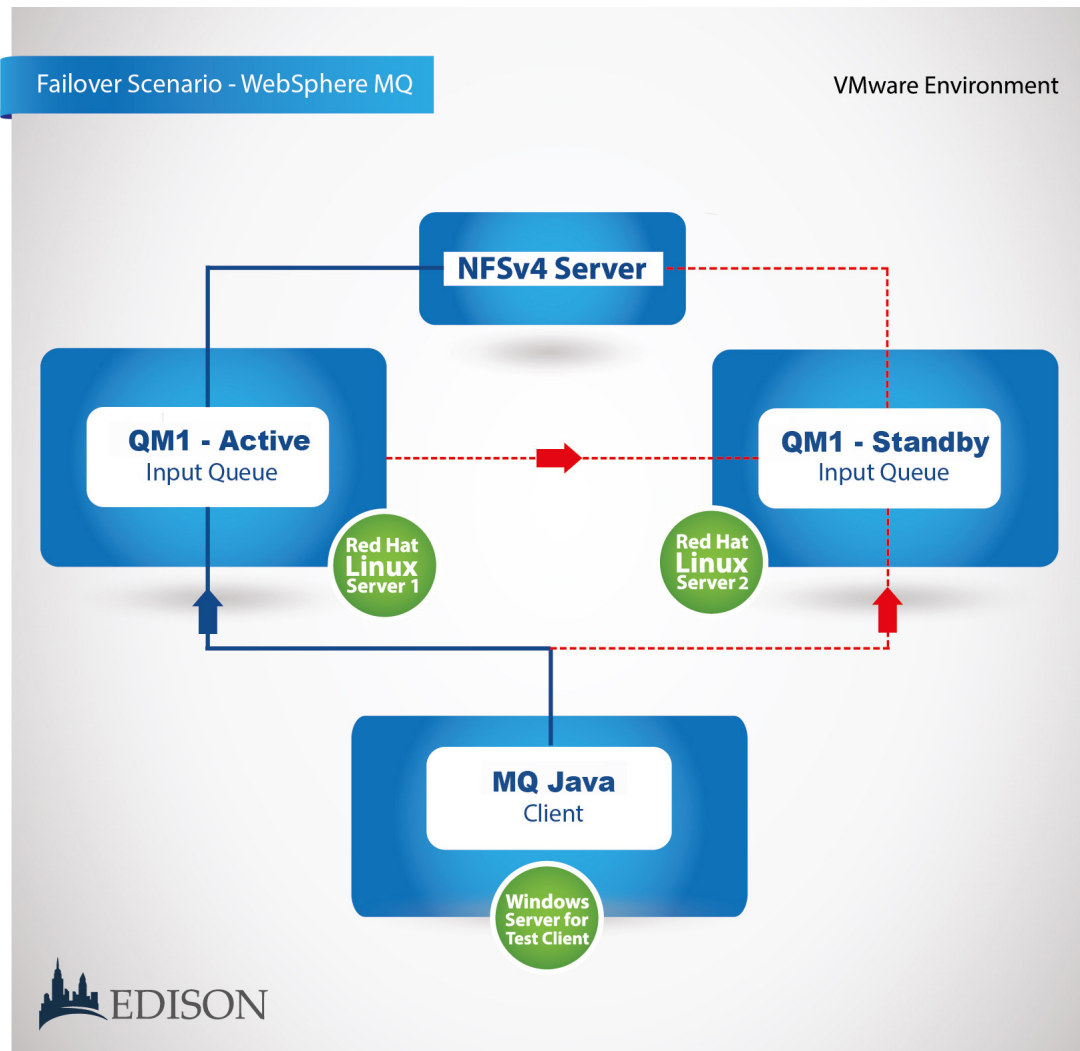
### WebSphere MQ Configuration #1: Active/Standby

- Shared File System (NFSv4)<sup>1</sup>
- Two WebSphere MQ servers configured with multi-instance queue managers<sup>2</sup>
  - One server running an active Queue Manager
  - One server with the Queue Manager on standby

<sup>1</sup> In Edison's test scenario, the HA/Failover capability around the Shared File System was not tested. In true production systems, this server, NAS, or other components are critical to ensure availability and redundancy.

<sup>2</sup> Documentation for creating and configuring multi-instance queue managers can be found at: [http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/index.jsp?topic=%2Fcom.ibm.mq.con.doc%2Fq018150\\_.htm](http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/index.jsp?topic=%2Fcom.ibm.mq.con.doc%2Fq018150_.htm)

*Note:* In WebSphere MQ terminology, “Multi-instance Active/Standby” is equivalent to Active MQ’s “Master-Slave” configuration.



**Figure 2: IBM WebSphere MQ 7.5 Test Failover Environment**

### ***Disruption Simulations***

For both systems, Edison sent and received messages to and from the Queue Manager (WMQ) or Broker (AMQ) via LoadRunner Java scripts. For WebSphere MQ, Edison used the standard MQ Java API (com.ibm.mq.\*).<sup>3</sup> For ActiveMQ, Edison used the Java API (org.apache.ActiveMQ.\*) and JMS.

<sup>3</sup> In addition to utilizing core APIs, Edison ran isolated tests using JMS and experienced similar testing results.

To simulate a network disruption and cause failover, Edison disconnected the network interface controller (NIC) using VMware vSphere Client while messages were being sent and received by the clients. To simulate a power outage, Edison powered down the Virtual Machine (VM).

Edison has produced detailed videos for a controlled failover scenario demonstrating how ActiveMQ and WMQ responded. The videos are available on YouTube at:

- **ActiveMQ:** <https://www.youtube.com/watch?v=ObFSQ38lq1k>
- **WebSphere MQ:** [https://www.youtube.com/watch?v=bS5r3\\_gkU9k](https://www.youtube.com/watch?v=bS5r3_gkU9k)

*Note:* The videos linked above do not present the complete testing results discussed in this white paper, but a simple failover scenario in a controlled environment: ActiveMQ's Master/Slave setup as compared with WebSphere MQ's multi-instance managers in a network disruption scenario (i.e., disabling the NIC in a VMware environment). For this test, Edison did not implement a "Network of Brokers" for ActiveMQ or "Clustering" for WMQ—a setup described in the "Management and Administration" section below.

In comparing the sent and received messages, Edison made the following discoveries:

#### ***"Power Outage": ActiveMQ Findings***

- As expected, the "slave" server became the "master" after the original master failed.
- During the actual failover, multiple messages were duplicated due to the client receiving messages originally from the master and subsequently on the new slave server after failover.

#### ***"Power Outage": WebSphere MQ Findings***

- No lost or duplicated messages.
- The client was notified about all messages that failed to transmit while the network was unavailable.

#### ***"Network Disruption": ActiveMQ Findings***

- When the NIC was restored on the original master, numerous messages were sent to the receiving client, configured to failover when not able to connect to its primary broker connection, in duplicate.
- In reviewing the web console, Edison identified numerous messages that were "sent" from the client machine, but were neither received after the restoration, nor logged as "failed to transmit."



- When the network connection to the original master was restored, it should have become the slave server, because the slave server became the active master and had a lock on the Shared File System. Instead, the original master also started up as an additional master server, gaining file lock from the original slave. The original slave remained in “master” status but no longer received messages from connected clients. In this configuration, subsequent failover would not have been possible—any failure of any one of the components would result in a complete loss of messaging service, despite the fact that network and hardware on the slave server was functioning properly.
- Restarting the VM cleared persistent message counts that were shown in the slave server’s web console after NIC was restored. Additionally, the master-slave configuration was not restored after the disruption. This is a known limitation of ActiveMQ.<sup>4</sup> Instead, there were two “master” brokers, with the reconnected master broker regaining the master file lock ownership, and the old master broker sending and receiving messages.
- In order to restore the master-slave configuration, the servers needed to be restarted. However, the system produced no warnings about stored messages; on restart, these messages were *lost*. To preserve these messages, an administrator would have to know there were available messages, manually back them up<sup>5</sup>, and restore them after the restart. This would be extremely difficult to manage in a live production system.

### ***“Network Disruption”: WebSphere MQ Findings***

- No lost or duplicated messages.
- The client was notified about all messages that failed to transmit during the disruption.

---

<sup>4</sup> Documentation for Pure Master Slave can be found at: <http://activemq.apache.org/pure-master-slave.html>

<sup>5</sup> For this paper, Edison did not evaluate ActiveMQ’s manual backup and restore capabilities.

# Transactional Integrity and Management

Edison used two scenarios to compare the native capabilities of WebSphere MQ and ActiveMQ. In both scenarios, Edison used a “Store and Forward” architecture, but in the first we used native APIs to manage transactions, and in the second we used WebSphere Application Server (WAS) v8.5.5. Whereas WebSphere MQ has a native Transaction Manager that allows us to ensure integrity across database servers, implementing “Store and Forward” for ActiveMQ, it required the creation of a “Network of Brokers” (as described in the “Management & Administration” section below).

While WebSphere MQ’s Transaction Manager allows users to administer their environment from a single console, an ActiveMQ “Network of Brokers” requires a separate console and editing of configuration files for each broker. As explained in the “Management & Administration” section below, this makes it far more difficult to administer an ActiveMQ system in an environment with more than a single server, not to mention a production environment where you have hundreds if not thousands of broker instances.

## *Scenario 1: Store and Forward - Using Native WebSphere MQ/ActiveMQ APIs*

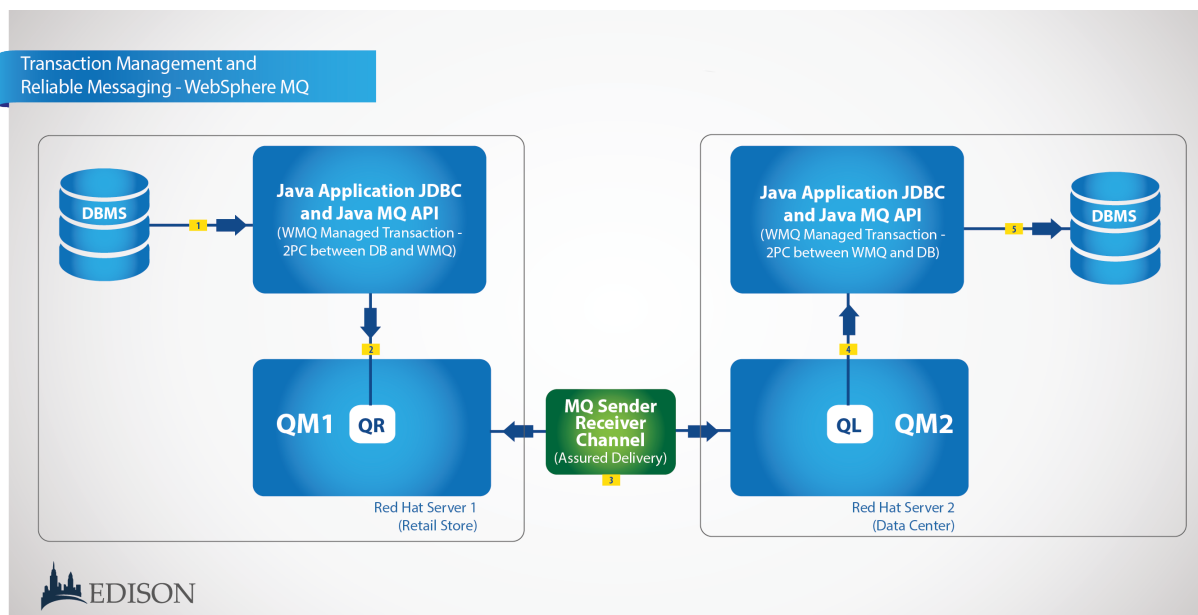
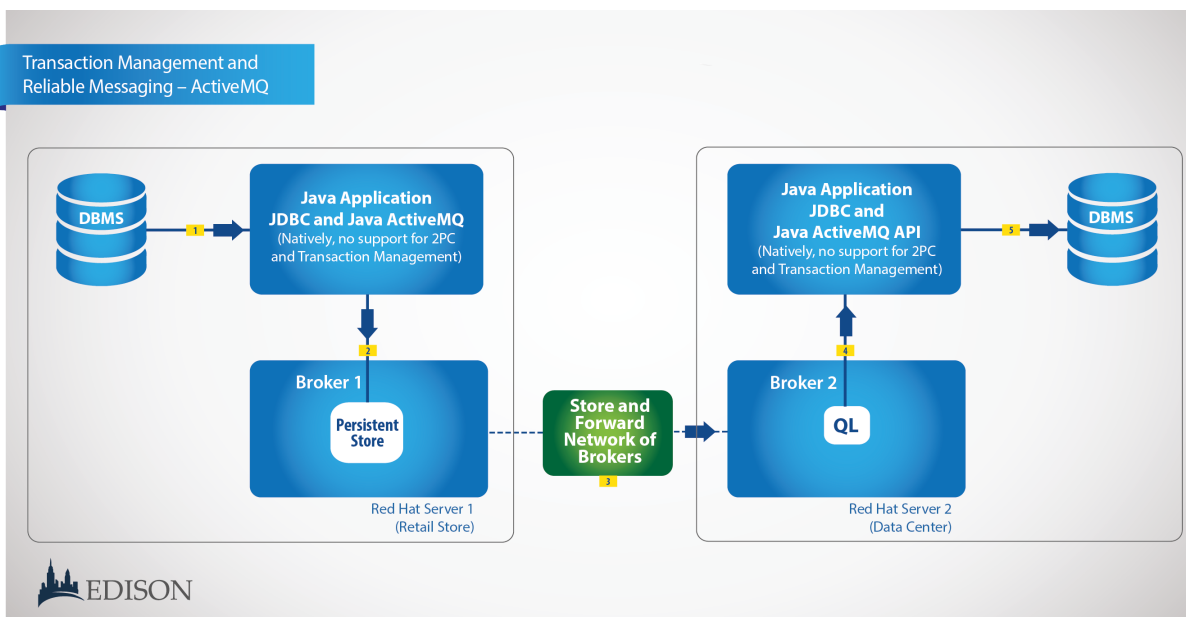


Figure 3: Store and Forward Using Native WebSphere MQ Transaction Manager



**Figure 4: Store and Forward Using “Network of Brokers” for ActiveMQ**

In the first scenario, Edison used the native capabilities of WebSphere MQ and a “Network of Brokers” for ActiveMQ. Edison ran transactions through in normal environments to establish a baseline and test for load balancing capabilities.

### Scenario Description:

- Client (i.e. retail store) reads data from the retail store database and stores messages on a queue “local to the retail store.” This process must be done under XA Transactional Control (2PC – Two Phase Commit) to ensure system integrity.
- Messaging infrastructure processes (gets) these messages “from the retail store server” and forwards (puts) them on to other Queue Manager and Broker in the “Datacenter,” thus accessing two or more independent Queue Managers (or Brokers). This step must assure “once and only once” delivery and no loss or duplication of messages is allowed.
- Client application running in the Data Center reads messages from the Data Center Queue and writes data into the Database located in the Data Center. This step also needs to be handled under XA Transactional Control (2PC – Two Phase Commit) to ensure system integrity.

Whereas WebSphere MQ's native Transaction Manager allows coordination of the database read/write and message put/get as one transaction (using 2PC), ActiveMQ does not provide Transaction Coordination across database and broker operations. In the scenario shown above, the transaction management capability of WMQ applies to both “In-Store” and “Data Center” client applications that move data between the database and the queue. The process of transferring the data between two WMQ Queue Managers

(one In-Store and one in Data Center) is done using Remote Queues, XMIT Queues, and client sender/receiver channels of WMQ to provide this reliable, assured delivering of messages. The In-Store client application is connecting and adding messages to the so-called “Remote Queue” defined in the In-Store QM. This allows client application to operate even when the network between the Store and Datacenter is not operational. All messages are kept in this Remote Queue on Store QM. Once network connection between the Store and Datacenter is available, the process of message processing is automatically performed by WMQ and all messages are reliably moved from the Remote Queue that is physically located In-Store to the Local Queue physically located in the Data Center. This is done in a way that network or hardware disruptions do not cause loss or duplication of messages; this is the same quality of service as a full 2PC transaction. After messages are moved from a Remote Queue to the Local Queue, the client application in the Data Center can consume these messages and write into the Data Center Database under the control of WMQ 2PC. All of the above means that the entire process of moving data from the In-Store Database to the Data Center Database is done asynchronously and reliably, with rollback and recovery in case of network, software or hardware failures.

ActiveMQ does not provide 2PC Transaction Manager and cannot coordinate 2PC transactions between the database and the Broker.

In this scenario, WebSphere MQ’s distributing queuing<sup>6</sup> with remote queues and sender and receiver channels and ActiveMQ’s Network of Brokers<sup>7</sup> were illustrated to perform the same function of delivering messages between server 1 and server 2. However, these components have architecturally different implementations and detailed analysis testing the reliability of Network of Brokers was not performed as part of these findings.

---

<sup>6</sup> Additional documentation on WMQ distributed queuing can be found at: [http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/index.jsp?topic=%2Fcom.ibm.mq.con.doc%2Fq015280\\_.htm](http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/index.jsp?topic=%2Fcom.ibm.mq.con.doc%2Fq015280_.htm)

<sup>7</sup> Additional documentation on ActiveMQ’s Network of Brokers and distributed queuing can be found at: <http://activemq.apache.org/how-do-distributed-queues-work.html>

## Scenario 2: Store and Forward - Leveraging WAS as Transaction Manager

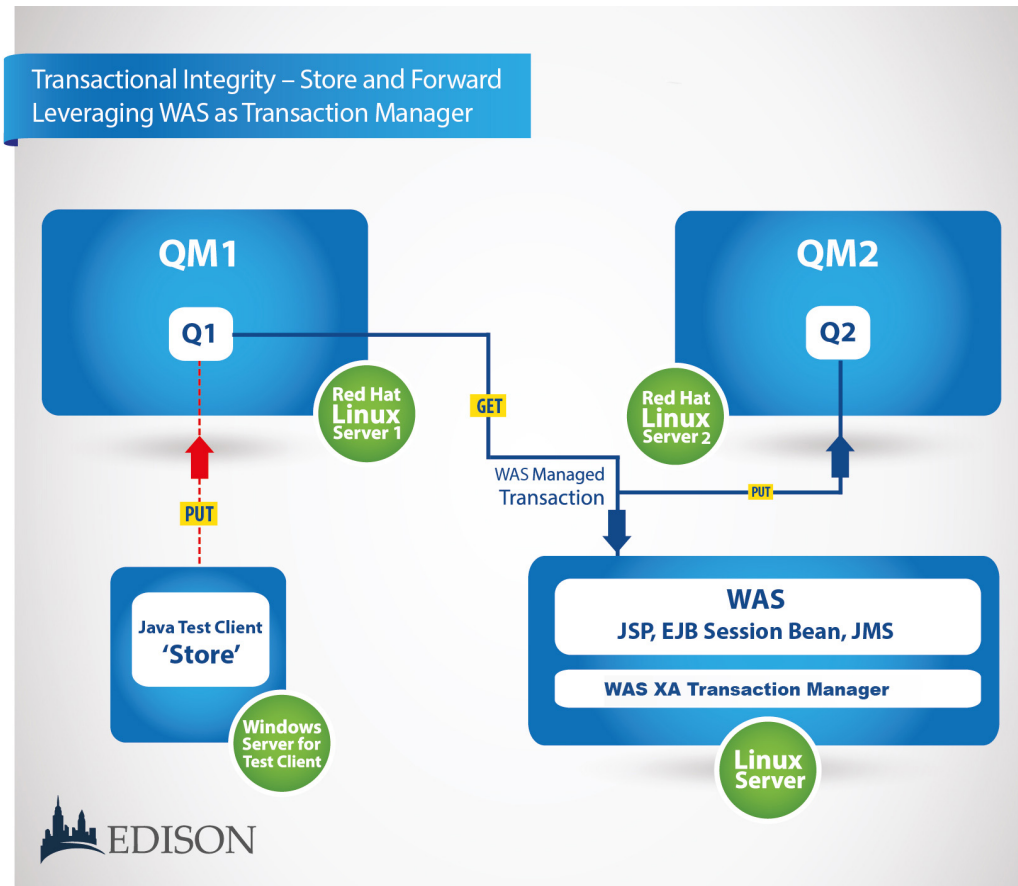


Figure 5: Scenario 2: Store and Forward Leveraging WAS as Transaction Manager

In the second scenario, Edison performed the same baseline and transaction failures, but used a WebSphere Application Server (WAS) server to manage the transaction.

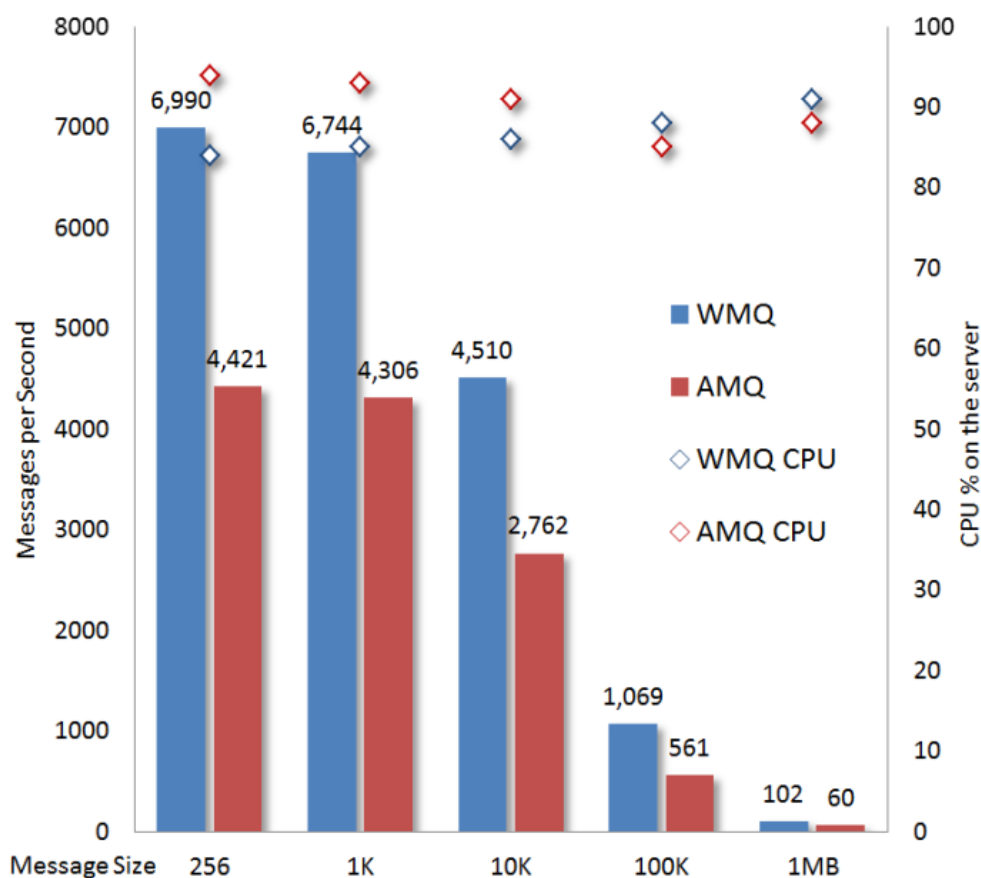
### Scenario Description:

- Client (Retail Store example) stores messages on a Server 1 queue.
- WAS server (hosted in Datacenter) processes these messages using JMS to receive message from Server 1 and send them to a queue on Server 2.
- The Session Bean EJB was doing both JMS receive and JMS send under 2PC Transactional Control of WAS to ensure integrity of the entire process.

Both systems were managed by WebSphere Application Server and did not lose or duplicate any messages under baseline conditions and during both types of failures.

## Performance

*Note:* Performance testing was not performed by Edison Group, but by IBM employee Roman Kharkovski.



**Figure 6: IBM WebSphere MQ vs ActiveMQ Performance Benchmark Results**

Performance results for persistent messaging were obtained through iterative tuning of WebSphere MQ and ActiveMQ. The best results for both products were produced at 80 concurrent client threads running IBM Performance Benchmark for JMS, running four instances of WebSphere MQ Queue Managers, and four instances of Apache ActiveMQ Brokers. Each instance had its own solid-state drive to which it writes data. Each 20-minute test run used one of five message sizes: 256 bytes, 1K, 10K, 100K, or 1MB. The measurements above are the average across six runs. On average, WebSphere MQ was 60 to 90 percent faster compared with ActiveMQ. Therefore, in order to achieve comparable results with ActiveMQ would require:

- 60 percent to 90 percent more hardware cost
- 60 percent to 90 percent more data center space
- 60 percent to 90 percent more cooling
- 60 percent to 90 percent more power
- 60 percent to 90 percent more software installed
- 60 percent to 90 percent more administration cost to manage it all

Full details of his testing methodology and results are available in the following posts on the [Why WebSphere? blog](#).

## Installation and Configuration

---

The installation procedures for both WebSphere MQ and ActiveMQ are straightforward processes: in both cases, installation was completed in a few minutes for the base install, with few errors or steps that needed to be repeated. With custom scripts – including one available on the [Why WebSphere? blog](#) for installing WMQ v7.5 on Red Hat Linux – WebSphere can be installed with one click, in 60 seconds. Similar scripts can be written for ActiveMQ.

There were, however, areas in which WebSphere MQ stood apart.

One key differentiator in the installation processes of WebSphere MQ and ActiveMQ is the availability and accuracy of documentation. While Edison was able to successfully install both products, the IBM documentation was much more thorough and accurate, including important details differentiating new installs from installations of new or later versions.<sup>8</sup>

ActiveMQ documentation was far less complete and reliable (often explicitly so, given the number of “TODO” tags on the [Active MQ 5.0 configuration docs](#)). In installing Active MQ, for instance, Apache’s documentation indicated that we should see the following message when the system is running properly:

```
"INFO ActiveMQ JMS Message Broker (ID:apple-s-Computer.local-51222-1140729837569-0:0) has started."
```

Instead, we saw the following:

```
"INFO: pidfile created : '/opt/media/apache-activemq-5.9.0/data/activemq-TTRH-vActiveMQ-2.pid' (pid '4046')."
```

In the absence of complete documentation, it is unclear what the consequences were for this instance of ActiveMQ. More importantly, WebSphere MQ is “certified” and supported on a much wider variety of operating environments than ActiveMQ. Additional comparisons of the two products’ system requirements and features can be found in the appendix.

With the exception of this issue, we encountered no significant problem with either installation performed on Red Hat Linux environments.

---

<sup>8</sup> There was one item missing from the WebSphere MQ documentation that system administrators should be aware of: the install will automatically create the “user mqm” and “group mqm” if they do not exist, but if they are not created manually, permissions will need to be set after the install.



## Management and Administration

---

For advanced users comfortable in command line environments on Windows or Linux, ActiveMQ and WebSphere MQ can be accessed by command shells, in Windows or Linux environments. However, the WebSphere MQ command shell allows users to conduct a variety of basic and advanced administrative tasks, such as changing maximum message sizes, setting message persistence and changing dozens of other configuration settings. With the ActiveMQ shell, users can only perform very simple administrative tasks like starting or stopping ActiveMQ brokers. For more advanced procedures, users need to edit files directly, and in many cases Active MQ does not support configuring common messaging properties (i.e. changing max message sizes or queue depths, which are controlled by available OS resources instead of an administrator being able to manage flexibly). And while both have fairly complete documentation, ActiveMQ documentation does have some gaps, which makes it difficult for all but the most advanced users to navigate. As one system administrator wrote on the ActiveMQ forums:

*"If you're going to be an ActiveMQ developer, you're going to have to get comfortable with their source code I'm afraid. The documentation is thorough, but not really exhaustive - almost every single class is documented, but the documentation doesn't explain everything. And the code quality... well, let's just say it isn't up to Microsoft standards."*

In contrast, the documentation available on [IBM's Infocenter](#) and in [IBM Redbooks](#) is complete and thorough, covering nearly all the configuration and administration topics that users might encounter while using WebSphere MQ.

For users who expect a graphical user interface (GUI), the contrast between the two products is just as great as it is for command line users. The ActiveMQ GUI console supports only the basic functionality, including visualization of the single server existing environment and the creation of new queues. For all other tasks—setting maximum message size, managing security, defining persistence, etc. —ActiveMQ, these are not available or users will have to go to the command line and file system and edit AMQ configuration files, or handle it through programming. ActiveMQ creates queues dynamically, as needed when the first producer or consumer connects to the destination. Therefore, ActiveMQ does not provide a command line capability to create queue. There is a possibility to define a queue in the XML configuration for the broker, which requires manual editing of XML file for every broker instance in the network.

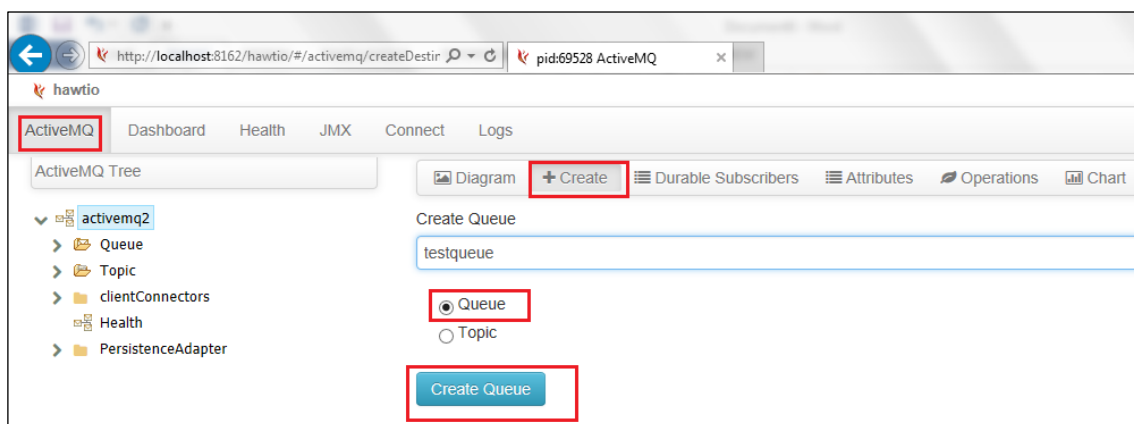
Moreover, WebSphere MQ Explorer allows administrators to make changes to numerous brokers from a single console. With ActiveMQ, each broker has a unique URL: if there are 100 brokers in an environment, 100 URLs will need to be opened and viewed to access them. This is true only of Master brokers: *ActiveMQ slave brokers do not have the admin console available; they can only be managed by the command line and XML configuration.*

The WebSphere MQ Explorer interface, unlike ActiveMQ, is also easy to use and provides a complete visibility of the environment. It also provides users with the ability to manage, customize, and administer detailed attributes on almost all aspects of the MQ environment (Queue Managers, Queues, Channels, Listeners, Topics, Publications, and Subscriptions, among others).

To provide a concrete example of the strengths and limitations of the two systems, Edison describes the step-by-step process of basic functionality in ActiveMQ and WebSphere MQ, including creating queues, setting sharing settings, assigning queue definitions, and managing multiple queues.

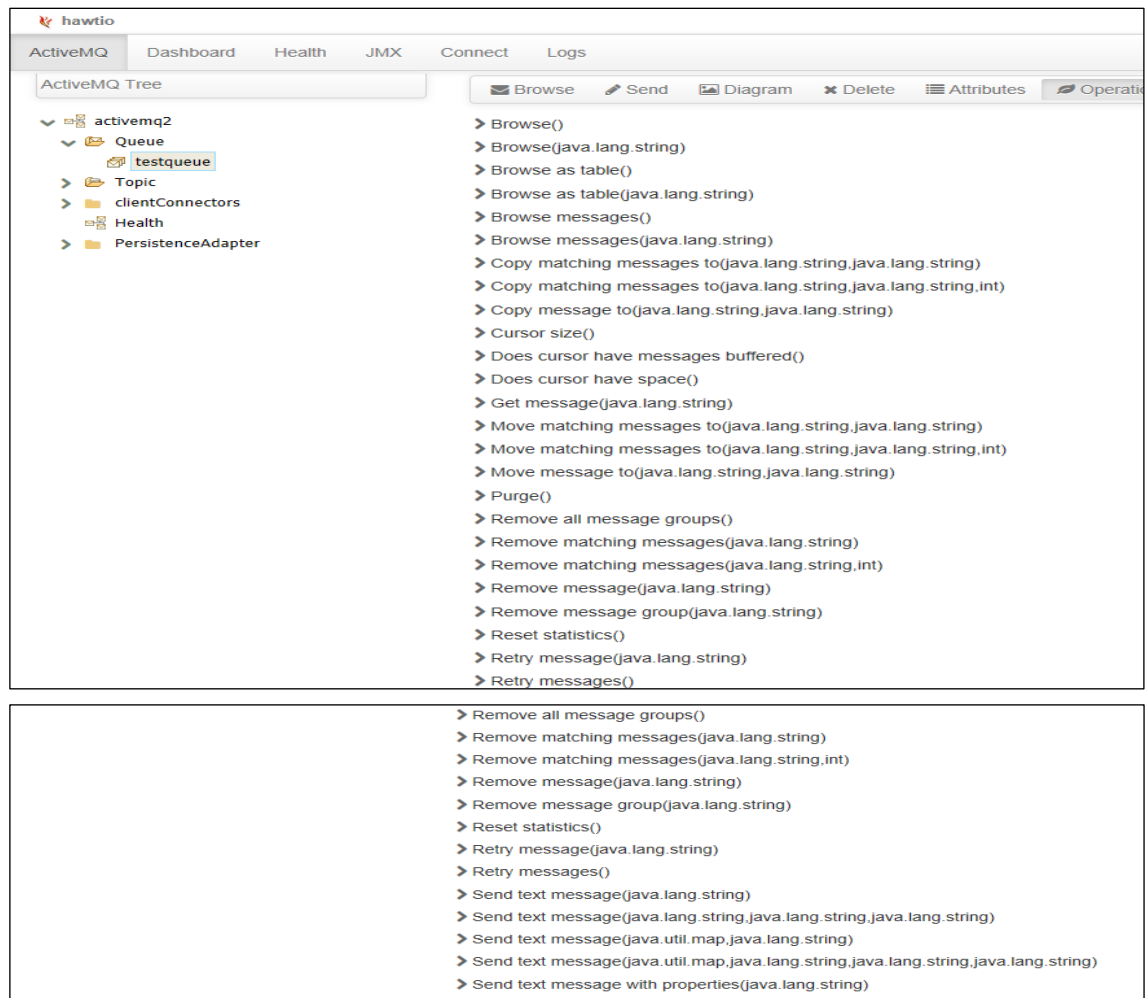
### ***Creating a Queue Via ActiveMQ Hawtio Console***

- Log into the hawtio console (<http://host:port/hawtio>)
- Click on “ActiveMQ”
- Click on “Create”
- Click on “Queues”
- Enter Queue Name and click on “Create Queue”



*Note:* Queues are stored on the file system/database. Users can configure XML for queues to be created on the start up to keep them persistent. Otherwise, if the queue is not there, it will recreate it next time the program tries to access it.

### *Creating a Queue Dynamically via JMX session and JNDI*

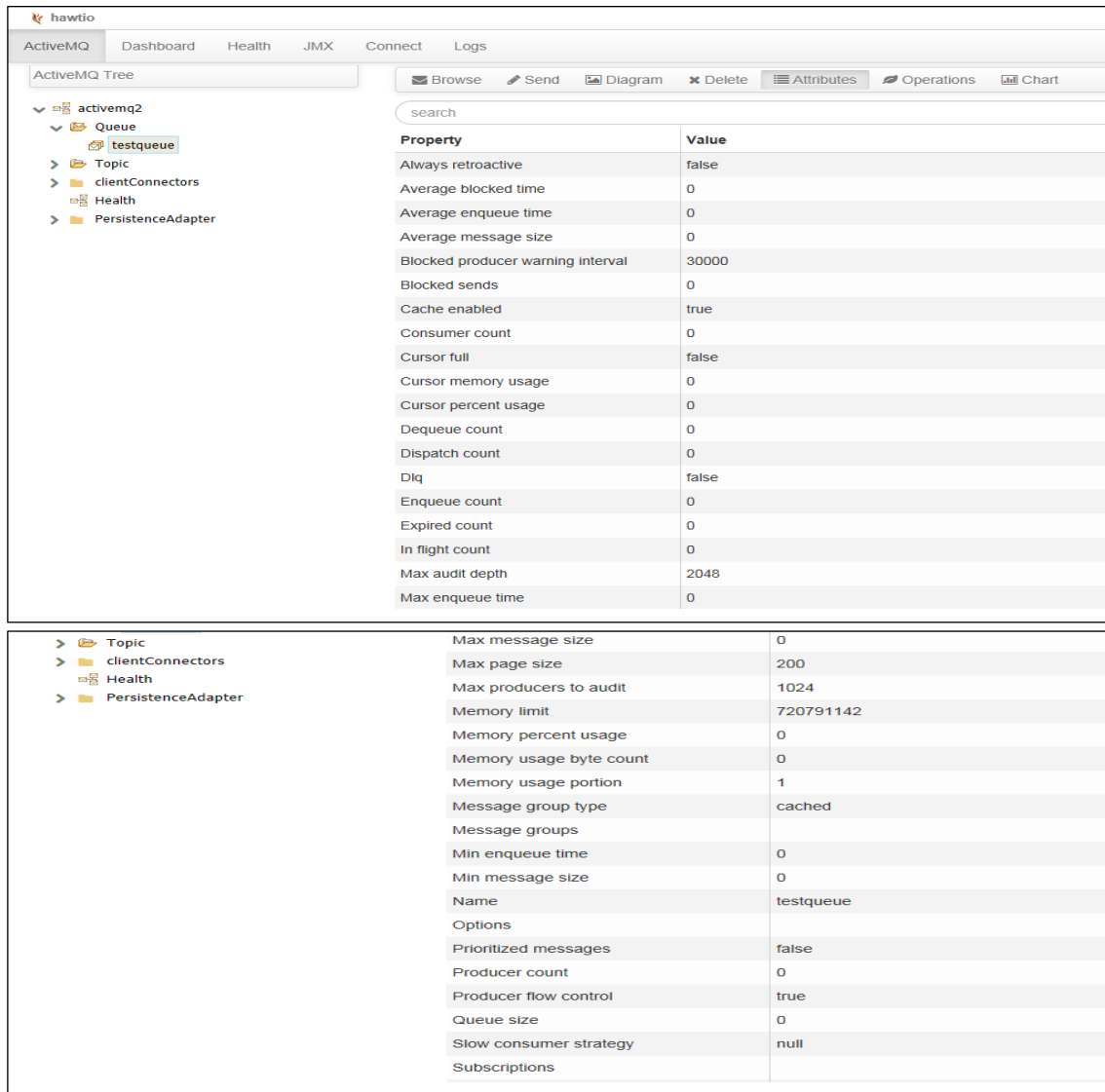


The screenshot shows the Hawtio ActiveMQ console. On the left, the 'ActiveMQ Tree' shows the hierarchy: `activemq2` > `Queue` > `testqueue`. The `testqueue` queue is selected. On the right, a list of actions is displayed, each preceded by a right-pointing arrow (>). The actions are:

- `Browse()`
- `Browse(java.lang.string)`
- `Browse as table()`
- `Browse as table(java.lang.string)`
- `Browse messages()`
- `Browse messages(java.lang.string)`
- `Copy matching messages to(java.lang.string,java.lang.string)`
- `Copy matching messages to(java.lang.string,java.lang.string,int)`
- `Copy message to(java.lang.string,java.lang.string)`
- `Cursor size()`
- `Does cursor have messages buffered()`
- `Does cursor have space()`
- `Get message(java.lang.string)`
- `Move matching messages to(java.lang.string,java.lang.string)`
- `Move matching messages to(java.lang.string,java.lang.string,int)`
- `Move message to(java.lang.string,java.lang.string)`
- `Purge()`
- `Remove all message groups()`
- `Remove matching messages(java.lang.string)`
- `Remove matching messages(java.lang.string,int)`
- `Remove message(java.lang.string)`
- `Remove message group(java.lang.string)`
- `Reset statistics()`
- `Retry message(java.lang.string)`
- `Retry messages()`
- `Remove all message groups()`
- `Remove matching messages(java.lang.string)`
- `Remove matching messages(java.lang.string,int)`
- `Remove message(java.lang.string)`
- `Remove message group(java.lang.string)`
- `Reset statistics()`
- `Retry message(java.lang.string)`
- `Retry messages()`
- `Send text message(java.lang.string)`
- `Send text message(java.lang.string,java.lang.string,java.lang.string)`
- `Send text message(java.util.map,java.lang.string)`
- `Send text message(java.util.map,java.lang.string,java.lang.string,java.lang.string)`
- `Send text message with properties(java.lang.string)`

Once the queue is created via the console above or dynamically, the user can perform a variety of actions on the queue e.g. browse, delete, send, purge messages (clear queue), and so forth.

The user can only view the attributes (i.e. read only mode).



The screenshot shows the Hawtio web console interface. The top navigation bar includes 'ActiveMQ', 'Dashboard', 'Health', 'JMX', 'Connect', and 'Logs'. The 'ActiveMQ Tree' on the left shows a hierarchy: 'activemq2' > 'Queue' > 'testqueue'. The main panel displays the 'Attributes' tab for 'testqueue'.

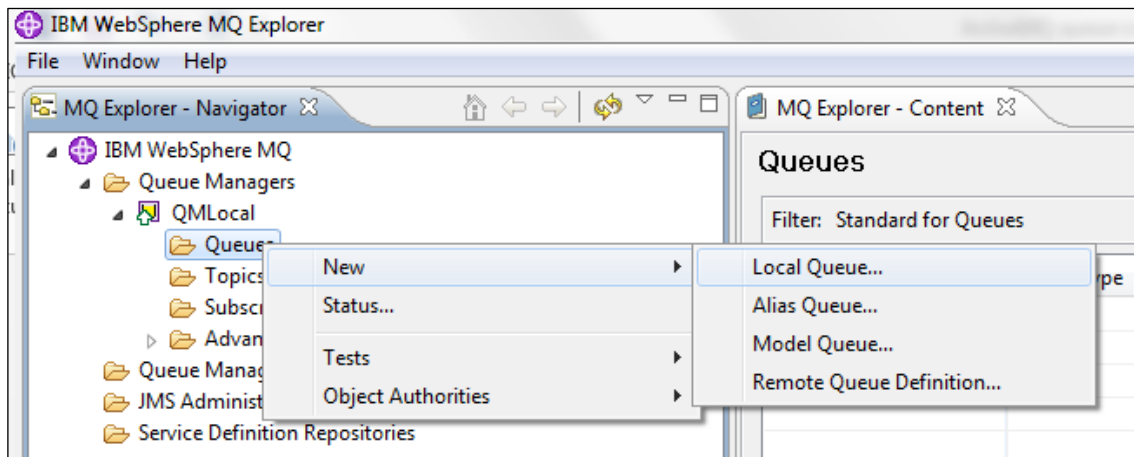
Property	Value
Always retroactive	false
Average blocked time	0
Average enqueue time	0
Average message size	0
Blocked producer warning interval	30000
Blocked sends	0
Cache enabled	true
Consumer count	0
Cursor full	false
Cursor memory usage	0
Cursor percent usage	0
Dequeue count	0
Dispatch count	0
DLQ	false
Enqueue count	0
Expired count	0
In flight count	0
Max audit depth	2048
Max enqueue time	0

Property	Value
Max message size	0
Max page size	200
Max producers to audit	1024
Memory limit	720791142
Memory percent usage	0
Memory usage byte count	0
Memory usage portion	1
Message group type	cached
Message groups	
Min enqueue time	0
Min message size	0
Name	testqueue
Options	
Prioritized messages	false
Producer count	0
Producer flow control	true
Queue size	0
Slow consumer strategy	null
Subscriptions	

### *Creating a Queue via WebSphere MQ Explorer*

- Select Queues in MQ Explorer - Navigator
- Select “New”
- Click on “Local Queue...”



MQ Explorer provides more useful queue attributes with customizations options that are not found in ActiveMQ web console, including the following:

- Put/Get permissions
- Message persistence
- Shareability
- Clustering
- Triggering
- Message Size and many others (see screenshots below).

In most cases, functionality that is unavailable in the ActiveMQ console can be handed by command line. However, there are many instances, such as creating unique messaging ports in environments with multiple brokers that require admins to *edit xml files directly* (see “Managing Multiple Queues” below).

## Managing Queue Properties in WMQ

New Local Queue

Change properties

Change the properties of the new Local Queue

General

Extended

Cluster

Triggering

Events

Storage

Statistics

General

Queue name: mqtestqueue

Queue type: Local

Description:

Put messages: Allowed

Get messages: Allowed

Default priority: 0

Default persistence: Not persistent

Scope: Queue manager

Usage: Normal

General

Extended

Cluster

Triggering

Events

Storage

Statistics

Extended

Max queue depth: 5000

Max message length: 4194304

Shareability: Shareable

Default input open option: Input shared

Message delivery sequence: Priority

Retention interval: 999999999

Definition type: Predefined

Distribution lists: Not Supported

Default read ahead: No

Default put response type: Synchronous

Property control: Compatibility

<ul style="list-style-type: none"> <li>General</li> <li>Extended</li> <li><b>Cluster</b></li> <li>Triggering</li> <li>Events</li> <li>Storage</li> <li>Statistics</li> </ul>	<h3>Cluster</h3> <div> <p>Sharing in Clusters</p> <p> <input checked="" type="radio"/> Not shared in a cluster         <input type="radio"/> Shared in cluster         <input type="radio"/> Shared in a list of clusters       </p> <p>         Default bind type: <span>On open</span> </p> <p>         CLWL queue rank: <span>0</span> </p> <p>         CLWL queue priority: <span>0</span> </p> <p>         CLWL use queue: <span>Queue manager</span> </p> </div>
<ul style="list-style-type: none"> <li>General</li> <li>Extended</li> <li>Cluster</li> <li><b>Triggering</b></li> <li>Events</li> <li>Storage</li> <li>Statistics</li> </ul>	<h3>Triggering</h3> <p>         Trigger control: <span>Off</span> </p> <p>         Trigger type: <span>First</span> </p> <p>         Trigger depth: <span>1</span> </p> <p>         Trigger message priority: <span>0</span> </p> <p>         Trigger data: <input type="text"/> </p> <p>         Initiation queue: <input type="text"/> <span>Select...</span> </p> <p>         Process name: <input type="text"/> </p>
<ul style="list-style-type: none"> <li>General</li> <li>Extended</li> <li>Cluster</li> <li>Triggering</li> <li><b>Events</b></li> <li>Storage</li> <li>Statistics</li> </ul>	<h3>Events</h3> <p>         Queue depth max events: <span>Enabled</span> </p> <p>         Queue depth high events: <span>Disabled</span> </p> <p>         Queue depth high limit: <span>80</span> </p> <p>         Queue depth low events: <span>Disabled</span> </p> <p>         Queue depth low limit: <span>20</span> </p> <p>         Queue service interval events: <span>None</span> </p> <p>         Queue service interval: <span>999999999</span> </p>

General Extended Cluster Triggering Events <b>Storage</b> Statistics	<b>Storage</b> Backout requeue queue: <input type="text"/> <input type="button" value="Select..."/> Backout threshold: <input type="text" value="0"/> Harden get backout: <input type="text" value="Hardened"/> NPM class: <input type="text" value="Normal"/>
General <b>Extended</b> Cluster Triggering Events Storage <b>Statistics</b>	<b>Statistics</b> Queue monitoring: <input type="text" value="Queue Manager"/> Queue statistics: <input type="text" value="Queue Manager"/> Queue accounting: <input type="text" value="Queue Manager"/>

### *Creating a Queue with runmqsc (WMQ Command line)*

Note: ActiveMQ does *not* support this functionality. Admins need to modify files manually in the file system.

- Utilize WMQ provided command line scripts as follows:

Command line to create a queue in WMQ:

```
DEFINE QLOCAL(queue)
8 : DEFINE QLOCAL(queue)
AMQ8006: WebSphere MQ queue created.
```

Available Queue attributes available while creating queues in WMQ:

```
DEFINE QLOCAL( q_name )
[ BOQNAME( string ) ]
[ CLUSNL( namelist_name ) ]
[ DEFBIND( NOTFIXED ; OPEN ) ]
[ DEFPRTY( integer ) ]
[ DEFREADA( NO ; YES ; DISABLED ) ]
[ DEFSOPT( EXCL ; SHARED ) ]
[ GET( ENABLED ; DISABLED ) ]
[ LIKE( qlocal_name ) ]
[ MAXMSGL( integer ) ]
[ HARDENBO ; NOHARDENBO ]
[ SHARE ; NOSHARE ]
[ PROCESS( string ) ]
[ PROPCCTL( COMPAT ; NONE ; ALL ; FORCE ) ]
[ QDEPTHHI( integer ) ]
[ QDPHIEU( ENABLED ; DISABLED ) ]
[ QDPMAXEU( ENABLED ; DISABLED ) ]
[ QSUCINT( integer ) ]
[ SCOPE( QMGR ; CELL ) ]
[ TRIGDPTH( integer ) ]
[ TRIGTYPE( FIRST ; EVERY ; DEPTH ; NONE ) ]
[ USAGE( NORMAL ; XMITQ ) ]
[ STATQ( QMGR ; ON ; OFF ) ]
[ MONQ( OFF ; QMGR ; LOW ; MEDIUM ; HIGH ) ]
[ CLWLKANK( integer ) ]
[ CLWLUSEQ( LOCAL ; ANY ; QMGR ) ]
[ BOTHRESH( integer ) ]
[ CLUSTER( cluster_name ) ]
[ DEFPRESP( SYNC ; ASYNC ) ]
[ DEFPERSIST( YES ; NO ) ]
[ DESCR( string ) ]
[ DISTL( YES ; NO ) ]
[ INITQ( string ) ]
[ MAXDEPTH( integer ) ]
[ MSGDLUSQ( PRIORITY ; FIFO ) ]
[ REPLACE ; NOREPLACE ]
[ TRIGGER ; NOTRIGGER ]
[ PUT( ENABLED ; DISABLED ) ]
[ QDEPTHLO( integer ) ]
[ QDPLOEU( ENABLED ; DISABLED ) ]
[ QSUCIEU( NONE ; HIGH ; OK ) ]
[ RETINTVL( integer ) ]
[ TRIGDATA( string ) ]
[ TRIGMPRI( integer ) ]
[ NPMCLASS( NORMAL ; HIGH ) ]
[ ACCTQ( QMGR ; ON ; OFF ) ]
[ CLWLPRTY( integer ) ]
```



Available queue attributes while altering the queue in WMQ:

```
ALTER QLOCAL< q_name >
[ BOQNAME< string > ]
[ CLUSNL< namelist_name > ]
[ DEFBIND< NOTFIXED ! OPEN > ]
[ DEFPRTY< integer > ]
[ DEFREADA< NO ! YES ! DISABLED > ]
[ DEFSSOPT< EXCL ! SHARED > ]
[ GET< ENABLED ! DISABLED > ]
[ INITQ< string > ]
[ MAXMSG< integer > ]
[ NPMCLASS< NORMAL ! HIGH > ]
[ PROPTCL< COMPAT ! NONE ! ALL ! FORCE > ]
[ PUT< ENABLED ! DISABLED > ]
[ QDEPTHLO< integer > ]
[ QDPLOEU< ENABLED ! DISABLED > ]
[ QSUCIEU< NONE ! HIGH ! OK > ]
[ RETINTVL< integer > ]
[ TRIGDATA< string > ]
[ TRIGGER ! NOTRIGGER ]
[ TRIGTYPE< FIRST ! EVERY ! DEPTH ! NONE > ]
[ USAGE< NORMAL ! XMITQ > ]
[ MONQ< QMGR ! OFF ! LOW ! MEDIUM ! HIGH > ]
[ SHARE ! NOSHARE ]
[ CLWLPRANK< integer > ]
[ CLWLUSEQ< LOCAL ! ANY ! QMGR > ]
[ BOTHRESH< integer > ]
[ CLUSTER< cluster_name > ]
[ DEFPRESP< SYNC ! ASYNC > ]
[ DEFPSIST< YES ! NO > ]
[ DESCR< string > ]
[ DISTL< YES ! NO > ]
[ HARDENBO ! NOHARDENBO ]
[ MAXDEPTH< integer > ]
[ MSGDLUSQ< PRIORITY ! FIFO > ]
[ PROCESS< string > ]
[ QDEPTHHI< integer > ]
[ QDPHIEU< ENABLED ! DISABLED > ]
[ QDPMAXEU< ENABLED ! DISABLED > ]
[ QSUCINT< integer > ]
[ SCOPE< QMGR ! CELL > ]
[ TRIGDPH< integer > ]
[ TRIGMPRI< integer > ]
[ ACCTQ< QMGR ! ON ! OFF > ]
[ STATQ< QMGR ! ON ! OFF > ]
[ CLWLPRTY< integer > ]
[ FORCE ]
```

Command line to clear the queue in WMQ:

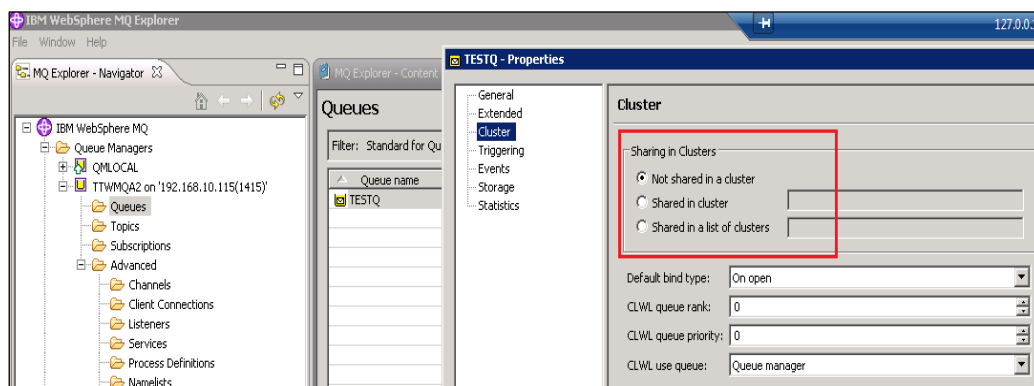
```
CLEAR QLOCAL<QUEUE>
16 : CLEAR QLOCAL<QUEUE>
AMQ8022: WebSphere MQ queue cleared.
```

Command line to delete the queue in WMQ:

```
DELETE QLOCAL< q_name >
[ PURGE ! NOPURGE ]
```

## Setting Queue Sharing Settings

WebSphere MQ easily defines whether a queue will be shared on a cluster(s):



In ActiveMQ, all queues on a broker are shared by default when two brokers are connected in a cluster. *This default setting cannot be changed.* The advantage of WebSphere MQ's functionality is twofold:

- It is easier to specify what queues are shared in each cluster
- Shared clusters provides out-of-the-box load balancing between brokers

In ActiveMQ, by contrast, admins need to manually edit configuration files to implement a "Network of Brokers." Though documentation on this option is very limited, Edison was able to implement a "Network of Brokers" using the following steps:

***Steps to set-up Network of Brokers (i.e., consumer needed on remote brokers to forward messages)***

- Go to the local broker config xml file i.e. /apache-activemq-5.9.0/conf/activemq.xml
- Add the following code on the broker:

```
<networkConnectors>

<networkConnector
uri="static://(tcp://remotehost1:61616,tcp://remotehost2:61616)"/>

</networkConnectors>
```

Where 61616 is the openwire transport connector on both remote hosts:

- Start both remotehost1 and remotehost2 brokers
- Start local broker

***Steps to set-up a Pure Network of Brokers (i.e., no consumer needed on remote brokers to forward messages)<sup>9</sup>***

- Go to the local broker config xml file i.e. /apache-activemq-5.9.0/conf/activemq.xml
- Add the following code on the broker:

```
<networkConnectors>
```

---

<sup>9</sup> For this paper, Edison did not evaluate the reliability or scalability of Network of Brokers or do a detail comparison of technical features, architecture, and capabilities of WebSphere MQ Clustering vs ActiveMQ Network of Brokers.

```
<networkConnector
uri="static://(tcp://remotehost1:61616,tcp://remotehost2:61616)"
staticBridge="true">

<staticallyIncludedDestinations>

<queue physicalName="queueName"/>

</staticallyIncludedDestinations>

</networkConnector>

</networkConnectors>
```

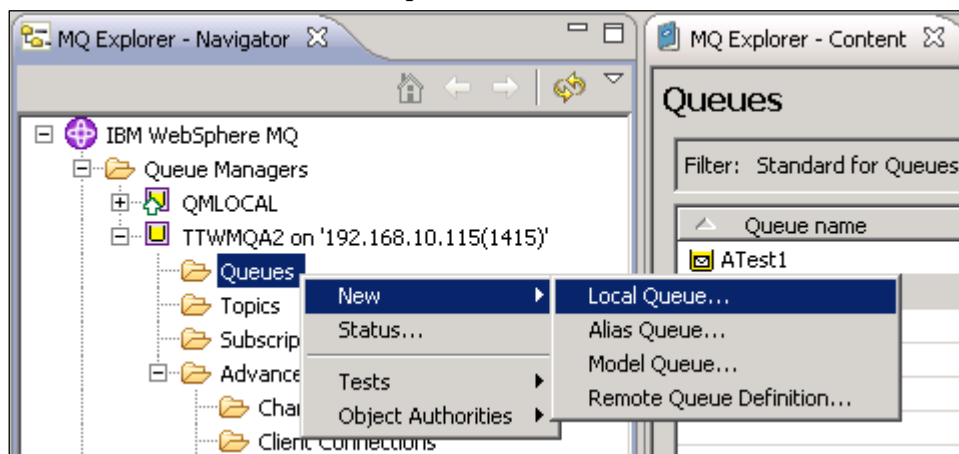
Where 61616 is the openwire transport connector on both remote hosts:

- Start both remotehost1 and remotehost2 brokers
- Start local broker

A simple test sending messages to the local host broker showed that the messages were load balanced and forwarded to both remote brokers.

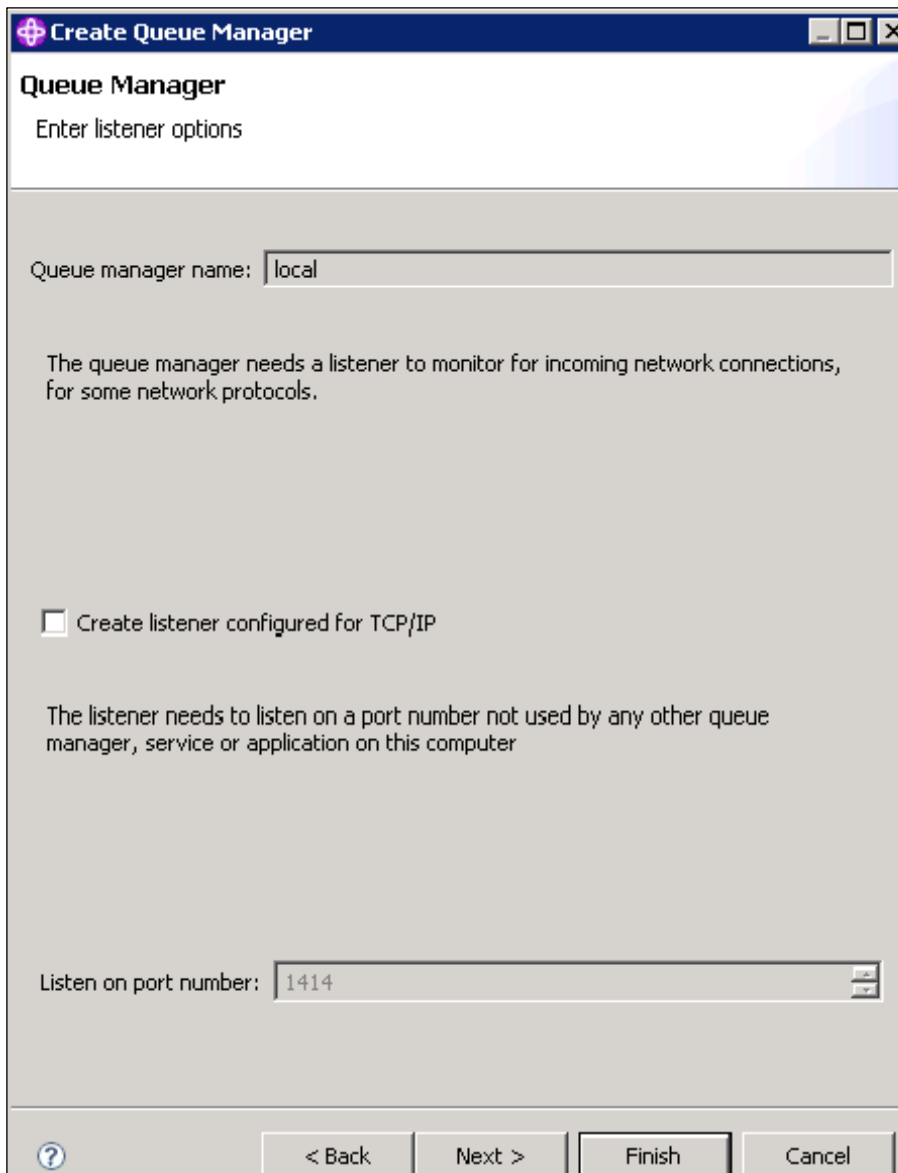
### *Assigning Queue Definitions*

WebSphere MQ allows you to define local, remote, alias and model queue definitions. ActiveMQ does not have this option.



## Managing Multiple Queues

With WebSphere MQ, multiple local queue managers can be created without the need for a port. However a listener port is required to monitor for incoming network connections.



The image shows a 'Create Queue Manager' dialog box with a blue title bar. The main area is titled 'Queue Manager' and contains the text 'Enter listener options'. There is a text field for 'Queue manager name:' with the value 'local'. Below this is a paragraph: 'The queue manager needs a listener to monitor for incoming network connections, for some network protocols.' There is an unchecked checkbox labeled 'Create listener configured for TCP/IP'. Below this is another paragraph: 'The listener needs to listen on a port number not used by any other queue manager, service or application on this computer'. At the bottom, there is a text field for 'Listen on port number:' with the value '1414'. The bottom of the dialog has a row of buttons: a help button (question mark in a circle), '< Back', 'Next >', 'Finish' (which is highlighted), and 'Cancel'.

In ActiveMQ, the transport connectors' ports (openwire, amqp, stomp, mqtt and ws) have to be unique for all the brokers on the same host in order for them to run concurrently. This has to be edited *manually* on the activemq.xml file. In a production environment with even a modest number of brokers, manually editing activemq.xml

files is both time consuming and – because of the absence of change logs and standard documentation – completely unscalable.

```
<transportConnectors>

<!-- DOS protection, limit concurrent connections to 1000 and frame
size to 100MB -->

<transportConnector name="openwire"
uri="tcp://0.0.0.0:62001?maximumConnections=1000&wireFormat.maxFrameSize=104857600"/>

<transportConnector name="amqp"
uri="amqp://0.0.0.0:5201?maximumConnections=1000&wireFormat.maxFrameSize=104857600"/>

<transportConnector name="stomp"
uri="stomp://0.0.0.0:62011?maximumConnections=1000&wireFormat.maxFrameSize=104857600"/>

<transportConnector name="mqtt"
uri="mqtt://0.0.0.0:2001?maximumConnections=1000&wireFormat.maxFrameSize=104857600"/>

<transportConnector name="ws"
uri="ws://0.0.0.0:61611?maximumConnections=1000&wireFormat.maxFrameSize=104857600"/>

</transportConnectors>
```

Also for each broker web console to be launched, the port has to be unique. This setting is changed on the jetty.xml file.

```
<bean id="jettyPort" class="org.apache.activemq.web.WebConsolePort"
init-method="start">

<!-- the default port number for the web console -->

<property name="port" value="8168"/>

</bean>
```

## Appendix 1: Feature Comparison

The table below<sup>10</sup> compares the availability of major features in IBM WebSphere MQ 7.5 and Apache ActiveMQ 5.9. Where workarounds and third-party solutions are available, Edison lists them in the notes to each feature.

Feature	WebSphere MQ	ActiveMQ
<b>Messaging</b>		
JMS 1.1, JMS 2.0	JMS 1.1 is fully supported  JMS 2.0 is fully supported with WebSphere MQ v8	JMS 1.1 support only
AMQP (Advanced Messaging Queue Protocol) support	Supported through a bridge by passing a JMS component out through an AMQP component	AQMP supported on v5.8+
Java, C++/C#, PHP clients	Supported	Supported
Managed File Transfer	Provided for a cost via WebSphere MQ MFT	Not provided
Troubleshooting	Diagnostic error messages for components	Limited information and documentation
<b>Quality of Service</b>		
Failover	No lost or duplicated messages	Messages are lost and duplicated in many cases
High availability	Proven high availability	Master-slave, has many issues and in some cases network failures result in two competing masters
Transaction Manager	Provided (2PC between QM and	Requires third party

<sup>10</sup> This table was generated in collaboration with the IBM WebSphere competitive team.

Feature	WebSphere MQ	ActiveMQ
(TMgr)	DBMS)	
Can serve as XA resource	Can be managed by external transaction manager	Can be managed by external transaction manager
Performance	Proven as “best in class”	60% to 90% slower than WebSphere MQ for persistent messages
<b>Administration</b>		
Management GUI	WebSphere MQ Explorer is very feature rich	Limited features  File editing is often required
Management CLI	Rich set of commands for management	Some commands are provided, but file editing is often required
Management API	Rich API for management	Limited set of JMX beans available
One pane management	Admins can manage all servers from one console or one command line	Each server must be managed individually (i.e. 100 servers mean 100 consoles)
Deployment patterns	Provided in IBM Smart Cloud Orchestrator, IBM PureApplication System, IBM SoftLayer, IBM BlueMix PaaS	Requires third party
<b>Security</b>		
Message encryption	Advanced message level security and data encryption (WMQ AMS add-on required)	Requires custom programming
Auditing and logging	Mostly provided but requires some administrative actions	File editing actions are not audited

Feature	WebSphere MQ	ActiveMQ
Heartbleed bug	Does not impact WebSphere MQ	Impacts ActiveMQ as it relies on Open SSL
Authentication and Authorization	Feature provided out of the box	Feature provided out of the box
<b>Telemetry (MQTT)</b>		
MQTT support	IBM was co-developer of MQTT protocol. MQ provides own client and broker and supports Eclipse PAHO project and other open source projects like Mosquitto	MQTT supported on v5.6+ but many bug reports.  Apollo version implements MQTT plugin.
Mobile messaging	Mobile and Device Messaging Client Pack to develop applications for mobile and other devices  Integrates with IBM Worklight	Supports lightweight messaging with MQTT for mobile devices
<b>Miscellaneous</b>		
Documentation	Detailed and accurate	Incomplete and not always accurate
Disk and memory footprint	650 MB disk  Under 1 GB RAM	70 MB disk  2+ GB RAM
Integration with DataPower	Fully integrated	Not supported
Platform support	Over 20 platforms supported	3rd party support for limited set of platforms
Installation time	Scripted install takes 60 sec	Scripted install takes 60 sec



## Appendix 2: Items Not Covered In This Paper

---

While this paper evaluated several important capabilities of WebSphere MQ and ActiveMQ, there are a number of other tests and comparisons that fell outside of the scope of this research due to time constraints. These include, but are not limited to:

- Security features
- Scalability, from the point of view of being able to support x number of publishers and subscribers (in case of durable and non-durable pubs and subs)
- In-depth Total Cost of Ownership (TCO) analysis
- Troubleshooting/debugging capabilities
- Backup and restore features
- In depth analysis of difference between WebSphere MQ channels, clustering, and transmission queues and ActiveMQ “Network of Brokers”
- Availability of skillset and training material in the market to be able to support the platforms at an enterprise level
- Installation of updates (e.g., fixpacks, hot fixes)
- Development experience
- Features of the two platforms for cloud deployment scenarios
- Support provided and ease of use with other Integration products (ESB, or EAI, or B2B or B2C, etc.) in the market
- Third party and vendor provided tools available to monitoring and support from community
- Performance of non-persistent messaging for point to point and pub sub
- Comparison of WebSphere MQ with vendor branded distributions of ActiveMQ, for example Red Hat JBoss A-MQ