Apache ActiveMQ and IBM MQ: High Availability and Administrative Analysis

July 2015

Prepared by:

zibisGroup
RESEARCH & ADVISORY

555 Legget Drive, Suite 304 Tower A, Ottawa, ON Canada K2K 2X3
Tel: 613-518-8006 www.zibisgroup.com
# TABLE OF CONTENTS

Executive Summary .................................................................................................................. 2  
Overview .................................................................................................................................. 2  
The Test Environment ............................................................................................................... 3  
  Physical/Virtual Specifications ............................................................................................... 3  
  Software Specifications ......................................................................................................... 4  
Examining The Administrative Experience ............................................................................. 7  
  Product Installation ................................................................................................................ 7  
  Configuration ......................................................................................................................... 8  
  Management of a Clustered Environment ............................................................................. 9  
  Run Time Administration and Monitoring .......................................................................... 10  
Reliability and Failover ........................................................................................................... 17  
  Failure Scenarios .................................................................................................................. 17  
    Scenario 1: Temporary Network Interruption .................................................................... 19  
    Scenario 2: Server Maintenance ....................................................................................... 20  
    Scenario 3: Server Failure ................................................................................................ 20  
    Scenario 4: Multiple Server Failure ................................................................................ 20  
Final Analysis ........................................................................................................................... 21  
Commissioned Research ........................................................................................................ 23  
About Zibis Group ................................................................................................................... 23
EXECUTIVE SUMMARY

This report provides the results of a reliability and high availability examination of the most recent and popular open source and commercial message oriented middleware solutions; namely Apache ActiveMQ 5.11.1 and IBM MQ 8.0. This included message reliability and service availability testing, with a look into the administrative experience with respect to productivity and overall capabilities.

A number of failover and reliability tests were completed using a clustered environment with four servers, each with an active service (i.e., Apache ActiveMQ Broker or IBM MQ Queue Manager) and a corresponding failover (i.e., Slave/Standby). These tests included a temporary network interruption, a predictable shutdown, and a catastrophic power failure.

The following outlines some high level observations discussed in greater detail throughout the report:

- Apache ActiveMQ passed testing when a Broker was fully shutdown (planned or unplanned). However, when encountering a temporary network interruption, it created a situation where there was message duplication and a potential for lost messages. Similarly, when using the “more reliable” JDBC as the persistence store, a separate unrelated issue resulted in a concern around lost messages. IBM MQ passed all testing without message duplication or potential for lost messages.
- Apache ActiveMQ provides no centralized way to manage a cluster (i.e., Network of Brokers), requiring administrators to access each individual Web-based console (i.e., a unique URL for each Broker); cumbersome in environments with a high number of Brokers. IBM MQ provides centralized management through either the CLI or the MQ Explorer GUI.
- The Apache ActiveMQ Web-console is not available for Slave brokers, meaning their status cannot be viewed other than checking that the process is actually running on the server.
- IBM MQ provides users with significant documentation. Unfortunately Apache ActiveMQ documentation was often outdated, which resulted in significant time spent searching or on more than one occasion resorting to online forums.
- IBM MQ can sometimes appear more complicated because of its plethora of feature functionality options. While Apache ActiveMQ may be simpler to configure initially, it does not have the same granular control over queues available through IBM MQ.
- Apache ActiveMQ provides a mix and match of management facilities forcing administrators to use and switch between interfaces. For example, some actions require manually editing XML configuration files, using the command line, or using the Web-based GUI. IBM MQ allows administrators to perform virtually all functions through the command line, the GUI, and the API; administrators are not forced to switch between them.
- IBM MQ supports automation through scripting since all actions can be completed through the command line.

OVERVIEW

The following provides an objective functional comparison of the most recent releases of Apache ActiveMQ (5.11.1) and IBM MQ (8.0) generally available at time of testing. IBM MQ is the most popular commercial message oriented middleware solution (based on market share reports) while Apache ActiveMQ currently has one of the largest number of installations of an open source message broker.

The following analysis is based on a series of hands-on tests that assessed how each solution performed from the perspectives of administration, high availability, and failover. Each environment was tuned and configured accordingly, and results were tracked with any differences and abnormalities documented.
The following includes, but is not limited to, the baseline project testing parameters for each of the offerings:

- **Administration Experience**
  - Installing and configuring a clustered environment of four independent servers.
  - Managing a distributed clustered environment (add nodes, start/stop cluster, remove nodes, perform maintenance, change configuration options, create/update/delete queues, clusters, topics, etc.).
  - Using a GUI, command line, and APIs to manage the environment.
  - Monitoring of workload, connected users, messages, errors, storage, and overall system health.

We reviewed and compared the administration experience with a focus on the above noted areas in addition to others that warrant further investigation and as determined over the duration of the testing.

- **Reliability and Failover**
  - Failover in a clustered environment (e.g., fail to backup server, restore master server, fail more than 1 server, etc.)
  - Availability during planned and unplanned outages (e.g., hardware and software failures, software and hardware upgrades and patches).
  - Checking for lost or duplicate messages.
  - Availability of the service during outages of different kinds.
  - Time to failover the service (client connection) and data failover (server messages moved to another server).

The objective here is not to provide a How To tutorial on the setup and configuration of the Apache ActiveMQ and IBM MQ environments, but rather provide a higher-level review with regard to the differences between the administrative and failover experiences.

**THE TEST ENVIRONMENT**

The following sections describe the software products and versions that were used for the tests. It covers the general software solutions (i.e., common across all solution implementations) first, and then moves on to the specific vendor solutions. For both vendors, an overview of the solution architecture and common terms are introduced to provide a baseline for comparison purposes.

**PHYSICAL/VIRTUAL SPECIFICATIONS**

To provide a real world type scenario, two physical servers were used to represent two physical racks within a data center. This allowed the creation of multiple virtual machines that represent the servers that would be separately installed in these two physical racks. This affords some redundancies should there be any failures related to either of the racks in the data center. More specifically, should connectivity to an entire rack be interrupted, the servers located on the separate rack theoretically could take over the workload.

The Physical Machines used for testing included:

- Server 1 – Generic with Intel Core i5 Quad Core; 2.3 GHz; 16 GB RAM; Gigabit NIC; Windows 7 Pro – 64bit
- Server 2 – HP Model PD 600 with Core i7 Quad Core; 3.4 GHz; 16 GB RAM; Gigabit NIC; Windows 7 Pro – 64bit
The virtual environment included the creation of five (5) virtual machines including:

- **HostNFS** (192.168.1.60) – used as the NFSv4 server and database server to provide persistent storage for respective vendor solutions.
- **Host11** (192.168.1.61) – representing Rack 1 Server 1
- **Host12** (192.168.1.62) – representing Rack 1 Server 2
- **Host21** (192.168.1.71) – representing Rack 2 Server 1
- **Host22** (192.168.1.72) – representing Rack 2 Server 2

### SOFTWARE SPECIFICATIONS

The following provides a brief introduction to the software solutions used for testing and also introduces the reader to the respective nomenclature for the covered technologies and techniques.

#### Messaging Solutions

The messaging solutions used in this study included **Apache ActiveMQ 5.11.1** and **IBM MQ 8.0**. Each solution was the most recent generally available and supported version at the time of testing.

**Apache ActiveMQ 5.11.1**

Apache ActiveMQ implements the Java Message Service 1.1 specification and offers features such as support for Enterprise Integration patterns, the Spring Framework, and transactions. For persistence of messages, different stores can be used, including: the default file-based KahaDB database on a shared file system; JDBC with a relational database (e.g., IBM DB2, Oracle, MySQL); and a Replicated LevelDB Store, which theoretically provides higher performance than KahaDB (although not officially proven) without the requirement to setup a highly available shared file system.

**ROADMAP NOTE:** Apache ActiveMQ Apollo is a recent development based on the experience of the ActiveMQ Project. In order to be “faster, more reliable, and easier to maintain” than ActiveMQ, a completely new architecture is being introduced. That architecture is based on the Scala programming language, which supports the development of concurrent systems. The threading of the Apollo broker differs fundamentally from that of the ActiveMQ solution. All tasks are performed asynchronously and non-blocking, which theoretically contributes to increased performance and stability. Although written in Scala, the broker can be used in a Java environment. There is no client library for Apollo itself. Therefore other clients can be used for the protocols supported by Apollo, which include MQTT, OpenWire, and STOMP. Apache ActiveMQ Apollo is noted here because it may have a significant impact on future development of the existing ActiveMQ. Organizations looking at implementing a messaging solution today, may need to keep this in mind given the concern that they may need to migrate between solutions in the near future.

The default KahaDB offers good performance with few resource requirements, and is optimized for messaging. It is easy to implement and is the recommended option when using a shared network file system. The use of JDBC as a persistent message store can be considered more reliable but is considerably slower. Finally, for “those willing to try out new tech,” LevelDB, while considered the fastest, introduces additional layers of implementation complexity.

**CAVEATS:** The LevelDB persistent store does not yet support XA Transactions or storing data associated with Delay and Schedule Message Delivery, which are stored in separate non-replicated KahaDB data files. Unexpected results will occur if using Delay and Schedule Message Delivery with a replicated LevelDB store since that data will not be there if/when a Master fails over to a Slave.

To provide failover in the event of catastrophic hardware failure, Apache ActiveMQ uses a Master/Slave model. Using this approach, Apache ActiveMQ can run as many Slave brokers as required. On startup, one Master grabs an exclusive lock on the broker file directory (when using a shared file system) or the shared...
broker database (if using JDBC). All other brokers are considered Slaves and pause waiting for an exclusive lock. If the Master loses its connection to the database or its exclusive lock then it immediately shuts down. If the Master shuts down or fails, one of the Slaves will grab the exclusivity lock and will be elevated to the Master position, starting all of its transport connectors and taking over any messages still in the persistent store (i.e., to prevent message loss). If the failed server is restarted, it will rejoin as a Slave waiting to become the Master should the new Master shutdown or fail. The Master broker is the only node that accepts client connections. The other nodes in Slave mode do not accept client connections. For example, a Master with three Slaves means those three servers would be sitting idle waiting for the Master to fail.

To address high availability needs, several ActiveMQ brokers (or Master/Slave pairs) can be connected in a “Network of Brokers”. In a Network of Brokers queues and topics (i.e., publish/subscribe) are virtual (i.e., a queue is available on all nodes and the routing takes place automatically). This allows a client to connect to any broker in the network and fail over to another broker if necessary, providing, from the client’s perspective, a highly available cluster of brokers.

**NOTE:** A Network of Brokers does not provide high availability of messages, only service availability. More specifically, if a broker fails in a Network of Brokers, the messages owned by that broker remain inside the broker’s persistent store until the broker comes back online. In cases where reliability of messages is required, a Master/Slave implementation is used. Additionally, total message ordering is not preserved with a Network of Brokers. Total ordering works with a single consumer but a network bridge introduces a second consumer. In addition, network bridge consumers forward messages via produce.send(...), meaning they go from the head of the queue on the forwarding broker to the tail of the queue on the target. If a single consumer moves between networked brokers, the total order may be preserved if all messages always follow the consumer but this can be difficult to guarantee with large message backlogs.

Clients use the Failover Transport to connect to the available brokers. The Failover Transport layers reconnect logic on top of any of the other transports (previously known as Reliable Transport), which randomly (by default) chooses one of the composite URIs specifying the servers and corresponding ports (e.g., failover:(tcp://primary:61616, tcp://secondary:61616)?randomize=false) and attempts to establish a connection to it. If it does not succeed or if it subsequently fails, a new connection is established to one of the other URIs in the list. To avoid the requirement of manually updating clients whenever changes are made, a “transportConnector” (edited in the Broker XML configuration file) provides options to allow the broker to update clients automatically with information about new brokers available for failover. More specifically, if “updateClusterClients” is enabled, then clients will only need to know about the first broker to connect to. If new brokers join, the client will automatically be updated with the additional URI of that broker to connect to in the event of a network or broker failure.

**SECURITY NOTE:** Since v5.6+, Apache ActiveMQ has supported a Discovery protocol that allows clients to automatically discover the brokers available for failover. The Discovery protocol works similar to the Failover transport with the exception that it uses a discovery agent to locate the list of URIs to connect to. To be able to use Discovery to find brokers, the brokers require that the multicast discovery agent be enabled on the broker. The issue however, is that when using auto discovery of brokers without (or compromised) authentication, an attacker may be able to present itself as a legitimate broker and subsequently be able to catch and/or manipulate all messages that run through it.

**IBM MQ 8.0**

IBM MQ, formerly IBM WebSphere MQ, provides customers with a robust message oriented middleware solution that delivers Universal Messaging with a broad set of offerings to meet enterprise-wide messaging needs. This includes connectivity for the Internet of Things and mobile devices. IBM MQ is the most popular commercial solution for messaging across 80 different platforms including Windows, Linux, and IBM Mainframe.

To address the high availability needs of the enterprise and provide reliability and robustness of message traffic, ensuring that a message is never lost, IBM MQ offers two software-based solutions (there is a third for the mainframe only); Multi-Instance Queue Managers and Queue Manager Clusters.
• **Queue Manager Clusters** – supports the grouping of two or more Queue Managers on one or more systems into a Cluster. This provides automatic interconnection between them and allows queues to be shared among them for load balancing and redundancy.

  **CLUSTER MANAGEMENT:** IBM MQ uses a repository to collect and distribute the information about the queue managers that are members of a cluster. This includes queue manager names, their locations, their channels, which queues they host, etc. A queue manager that holds a complete set of information about every queue manager in the cluster has a full repository, where a partial repository contains information about only those queue managers with which it needs to exchange messages. The use of a cluster repository eliminates the need to manually update connecting client applications with network and cluster changes, and does not rely on multicast for discovery.

• **Multi-Instance Queue Managers** – are implemented in Active/Standby pairs with data residing on shared storage. By starting multiple instances, one instance becomes the Active Queue Manager and the other instance becomes a Standby. Should the Active Queue Manager fail, the Standby running on a different system will automatically take over.

**TL; DR:** Apache ActiveMQ and IBM MQ provide similar functionality from a messaging and service high availability perspective, simply with different nomenclature. More specifically, where IBM MQ provides scalability and service availability via a cluster of Queue Managers, Apache ActiveMQ uses the term Network of Brokers. Similarly, for failover, IBM MQ uses the terms Active/Standby while ActiveMQ uses the terms Master/Slave. There is one notable difference here in that ActiveMQ can have multiple Slave systems; IBM Multi-Instance Queue Managers are currently limited to a single Standby system.

**Virtualization Technology**

Each server (i.e., Host11, Host12, Host21, etc.) and subsequently the respective vendor message solutions were deployed on a virtual machine created using Oracle VirtualBox. While VMware hypervisor technology could have just as easily been used, Oracle VirtualBox was selected based on its availability through a GNU General Public License V2. The version used for this study was 4.3.28, available at http://download.virtualbox.org/virtualbox/4.3.28/VirtualBox-4.3.28-100309-Win.exe.

**Server Operating Environment**

All five servers were created using the CentOS Linux, a community-supported distribution of Red Hat Enterprise Linux (Red Hat only released source code per the open source license but did not provide compiled versions itself). In 2014, Red Hat announced that it was joining forces with CentOS to provide a more open roadmap and accelerate innovation. In short, it essentially provides users with a free version of Red Hat Enterprise Linux.

Similar to the selected messaging solutions, the latest supported generally available version of CentOS was used, more specifically, CentOS 7, x86_64.

**Test Tools**

To send and receive messages during testing, the Apache JMeter application was used as the dedicated test client. Apache JMeter is an open source Java application designed to load test functional behavior and measure performance. It was originally designed for testing Web Applications via HTTP but has since expanded to other test functions such as FTP, Database via JDBC, Message-oriented Middleware via JMS and more. While JMeter is available through the Apache Foundation, it provides a generic solution that requires the messaging vendor specific APIs to be loaded, making it suitable for testing across different vendor solutions.

JMeter version 2.13 (http://apache.mirror.vexxhost.com//jmeter/binaries/apache-jmeter-2.13.zip) was used for these tests, running from a desktop system with Microsoft Windows 7.
For testing the Request-Response capabilities of the vendor solutions, a “Server Side” replier was required to answer the request and send the response. In short, since the messaging solutions themselves are being tested, as long as the replier can predictably respond to every message it receives, the source and implementation are not overly important. As such, generic code was used.

EXAMINING THE ADMINISTRATIVE EXPERIENCE

PRODUCT INSTALLATION

To meet the availability requirements, Apache ActiveMQ Master/Server pairs with failover to one another were arranged in a Network of Brokers. Figure 1 below provides a visual representation of the implemented architecture. The arrangement of the Master/Slave failover pairs should be noted. Rather than Host11 and Host12 failing over to each other, Host11 on Server 1 and Host 22 on Server 2 are the first Master/Slave pair and Host12 and Host21 are the second Master/Slave pair. This was done to locate the corresponding Slaves on a separate “rack” from the Master in the event of multiple failures where an entire rack loses communication.

Similarly Figure 2 below provides a visual representation of the implemented IBM MQ architecture.
Overall, both messaging solutions prove relatively easy to install.

Apache ActiveMQ in summary requires the installation of a Java Runtime Environment, the creation of a user and group, and the expansion of the downloaded tar file. It provides a very easy installation process and uses the same installation package for both Linux and Microsoft Windows platforms.

IBM MQ can be installed via RPM (depending on the Linux distribution) or through a downloadable tar file. Compared to ActiveMQ, the installation automatically created the necessary user and group, and the IBM MQ installation packages are more specific to the platform (i.e., separate Windows installer).

While the installation process was not officially timed, Table 1 provides some insight into the download size and the average time required to install and configure the respective offerings for the implemented architecture described above (assuming all configuration options are known without requiring reference).

<table>
<thead>
<tr>
<th>Task</th>
<th>Apache ActiveMQ</th>
<th>IBM MQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Download Size</td>
<td>41.4MB + 91.5MB JRE</td>
<td>552MB (includes JRE)</td>
</tr>
<tr>
<td>Installation and Configuration Time</td>
<td>Approximately 2 hours</td>
<td>Approximately 4 hours</td>
</tr>
</tbody>
</table>

**CONFIGURATION**

For the configuration of Apache ActiveMQ, administrators are required to create a broker.xml file that contains the configuration for each message broker. Users can reuse the same broker.xml file to create brokers on multiple machines. In our case however, failover creates a scenario where two Master brokers could be running on the same server, which required an additional XML configuration file for the Slaves to
make sure the ports used by the Web-based Administrative Console (and the brokers themselves) were distinct. In short, there were a total of eight XML configuration files spread across four servers.

When configuring Apache ActiveMQ, unfortunately considerable time was spent researching and piecing together all the required details. More specifically, the online documentation on the Apache website is sometimes incomplete and not entirely up to date, requiring the need to resort to forums on more than one occasion.

**AUTHENTICATION AND AUTHORIZATION:** To ease failover and reliability testing in the implemented environment, authentication and authorization was not enabled (or in the case of IBM MQ, disabled); not recommended in a production environment. This included both Client-to-Broker (or Queue Manager) and Broker-to-Broker authentication. For example, in Apache ActiveMQ, if authentication is enabled for a Broker (either using simple XML configuration or JAAS), then other Brokers that wish to connect to it (e.g., in a Network of Brokers) must provide the proper authentication credentials via their corresponding `<networkConnector>` element.

IBM provides customers with extensive detailed documentation, providing the necessary configuration related information on the IBM website. This proved helpful in that the IBM MQ configuration provides significantly more feature functionality and flexibility than Apache ActiveMQ.

**FEATURES AND FLEXIBILITY:** By comparison, Apache ActiveMQ facilitates easy configuration, but lacks the granular control provided by IBM MQ. In general, with more flexibility (i.e., options and control) comes additional complexity. For example, with Apache ActiveMQ when a broker is in a “Network of Brokers”, all its queues and topics are shared with other brokers in the “Network of Brokers”. More specifically, there is no way to configure a private queue or topic to provide message flow isolation. While IBM MQ comparatively would require very little additional effort to create a queue on each server, it enables administrators to explicitly control queue placement to specific machines; helpful when dividing workload between machines of varying resource capacity.

**MANAGEMENT OF A CLUSTERED ENVIRONMENT**

Both messaging solutions can provide support for the easy addition and removal of nodes to and from a cluster on the fly.

The starting and stopping of a broker in Apache ActiveMQ can only be performed through the command line. When starting a broker, the corresponding XML file must be specified for the configuration of the broker if the default one is not used. With respect to adding and removing nodes, a new node can be added to an existing ActiveMQ cluster only when the Network of Brokers is configured to use Dynamic discovery. Otherwise with a Static configuration, all respective XML configuration files need to be updated and the brokers restarted; highly undesirable in a production environment.

**MULTICAST CONCERNS:** In the case of Apache ActiveMQ, while Dynamic connections (versus Static connections) support easily adding and removing brokers to and from a Network of Brokers, this functionality uses multicast, which may be blocked or unsupported in some environments. For example, many third party cloud platforms such as Amazon’s Virtual Private Cloud (VPC) do not directly support multicast between servers. This means that Static connections must be defined, and the currently running brokers will need to be systematically stopped and restarted with the updated XML configuration files. By comparison, IBM MQ does not require the use of multicast and does not require other members in the cluster to be restarted with the addition or subtraction of nodes.

Further, Apache ActiveMQ provides no central point of control over the cluster (i.e., Network of Brokers). If any broker in the cluster is started then the cluster is considered started with service. Conversely, the cluster is only considered stopped when all members of the cluster are stopped.

In IBM MQ, a member of a cluster can easily be added or removed at any time without a requirement to restart any other members in the cluster. Administrators simply specify the cluster the new member belongs to, and configures the connection channel to the cluster in the repository manager. To remove a member, administrators simply redefine the cluster send/receive channel. In short, IBM MQ clustering removes the need for complex configuration management.
CENTRALIZED MANAGEMENT: In short, Apache ActiveMQ does not provide a centralized way of managing the nodes in a Network of Brokers through the command line or through its Web-based Admin Console. The Apache ActiveMQ Web-based Admin Console is specific to each broker; meaning that in our environment there were eight (8) separately used ActiveMQ Admin Consoles. While IBM MQ does not support creating and starting Queue Managers remotely, it does allow remote administration to provide centralized control from the command line of a single server. Similarly, IBM MQ Explorer on a desktop can be configured with the credentials for each IBM MQ Server allowing administrators to manage multiple Queue Managers through a single MQ Explorer window.

Finally, in both solution cases, the only control an administrator has over which instance of a broker or queue manager becomes active first (i.e., Master or Active respectively) is the order in which they are started, with the first instance, in the case of Apache ActiveMQ, acquiring the lock to the data first and becoming active. However, while Apache ActiveMQ requires administrators to stop brokers to rebalance them across servers (i.e., so two active brokers are not on the same server), IBM MQ provides the facilities to force (i.e., “Switchover”) the Standby instance to take over as the Active instance without restarting the Queue Managers.

RUN TIME ADMINISTRATION AND MONITORING

In short, Apache ActiveMQ provides only basic administration monitoring methods, such as queue depth, connections, subscribers, and network connections. IBM MQ provides more comprehensive details such as queue route, channel status and much more.

Beyond the minimal command line capabilities provided by Apache ActiveMQ, it provides a rudimentary Web-based administrative GUI (that uses the open source “Jetty” Servlet Engine, defined in the broker’s XML configuration file) for actions such as creating queues and topics, sending messages, and browsing queues. Each ActiveMQ Web-based Admin Console is specific to a broker meaning that each broker requires its own browser window, a significant issue in an environment where 25 or more brokers are running.

Further, the ActiveMQ Web-based Admin Console is only available for Master brokers. This means that the Web console is not available for Slave brokers as they are technically not fully invoked until a lock on the persistence store can be obtained and subsequently elevated to the Master position. This means that there is no way for an administrator to graphically monitor or manage Slave brokers, and their status can only be checked based on whether or not the process is running on the server.

Figure 3 through Figure 6 provide a view of the Apache ActiveMQ Web-based Admin Console.
Figure 3: Apache ActiveMQ Jetty Administration Console

Figure 4: ActiveMQ Administration Queue Operations
MIX AND MATCH: Apache ActiveMQ provides a mix and match of administration through the CLI, the Web-based GUI, and API. Further, methods for managing Apache ActiveMQ appear to come and go. For example, a Command Agent, removed in version 5.9, allowed communication (to perform administration queries and commands) through the message bus itself to list available queues, topics, subscriptions, view metadata, browse queues, etc. This also
allows the use of XMPP to talk to the broker. Another method to retrieve stats for the broker is to send an empty message to the destination named ActiveMQ.Statistics.Broker along with a replyTo header. Using this method users can also retrieve statistics on all queue and topic subscriptions (i.e., send an empty message to the destination named ActiveMQ.Statistics.Subscriptions along with a replyTo header).

Some third party software can also connect to ActiveMQ to provide administration and monitoring functions on ActiveMQ, however these are essentially limited to the same functionality as the default Jetty Web-based ActiveMQ Administration Console (using JMX). For example, there are a number of external Web consoles available through separate open source projects (which vary in maturity) such as hawtio, RHQ (an operational monitoring tool which has support for Apache Camel along with other Apache projects like Tomcat, httpd, etc.), and HermesJMS, an extensible console that helps interact with JMS providers (i.e., not just ActiveMQ) to publish and edits messages, browse or search queues and topics, copy messages around and delete them.

IBM provides customers with several ways to administer and configure IBM MQ including: IBM MQ control commands, IBM MQ Script (MQSC) commands, Programmable Command Formats (PCFs), and IBM MQ Explorer.

Unlike Apache ActiveMQ, IBM MQ allows virtually all administrative and monitoring functions to be completed through either the command line or its MQ Explorer graphical user interface (or the API). It does not split capabilities between the CLI and the GUI, which would otherwise force administrators to use both. Instead, IBM MQ allows administrators to select the interface that best suits their needs. This also means that scripting can be used to automate a number of processes and tasks.

IBM MQ Explorer provides a more comprehensive GUI where users can simply click on an object to see detailed information or right click to update the object's properties. Users can create new Queue Managers, define channels, create clusters, and any other function that is available through the command line interface. Further, IBM MQ Explorer can be configured with the necessary authentication information to allow administration and management of the entire IBM MQ environment through a single MQ Explorer window.

While Figure 7 below provides a single view of the MQSC command line interface, Figure 8 through Figure 13 provide a high level view into the rich IBM MQ Explorer graphical user interface.

![Figure 7: MQSC Command Interface](image-url)
Figure 8: MQ Explorer Interface

Figure 9: IBM MQ Create Queue Manager
Figure 10: IBM MQ Queue

Figure 11: IBM MQ Queue Manager
Figure 12: IBM MQ Cluster Management

Figure 13: IBM MQ Explorer Check Object Properties
Table 2 provides a high level summary of the administrative and monitoring capabilities that are provided by the respective solution interfaces. What is important to note is that IBM MQ allows virtually all tasks to be performed through each available interface, while Apache ActiveMQ provides little consistency across the interfaces, and in some cases may require the editing of XML configuration files.

<table>
<thead>
<tr>
<th>Task</th>
<th>Apache ActiveMQ</th>
<th>IBM MQ</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Edit Config</td>
<td>CLI</td>
</tr>
<tr>
<td>Create Broker/QM</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Start Broker/QM</td>
<td>N/A</td>
<td>*</td>
</tr>
<tr>
<td>List Existing Brokers/QMs</td>
<td>N/A</td>
<td>*</td>
</tr>
<tr>
<td>Create Queue/Topic</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Monitor Queue/Topic</td>
<td>N/A</td>
<td>*</td>
</tr>
<tr>
<td>Manipulate Queue/Topic</td>
<td>N/A</td>
<td>N</td>
</tr>
<tr>
<td>Add/Remove Cluster Member</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Add New Connection Port</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Check Cluster Connections</td>
<td>N/A</td>
<td>N</td>
</tr>
<tr>
<td>Check Client Connections</td>
<td>N/A</td>
<td>*</td>
</tr>
</tbody>
</table>

(*) The Apache ActiveMQ community does provide some shell scripts in an attempt to add to the command line facilities but some of these capabilities are limited by platform or return information specific to predefined queries in the scripts (i.e., customization requires editing the scripts). In most cases, JMX must be enabled to perform any of the functions provided by the Command Line Tools.

To provide a more extensive view of the administrative experience of Apache ActiveMQ and IBM MQ, corresponding demo videos are available at:

- Apache ActiveMQ Administrative Demo – https://youtu.be/BMqfENuXOac
- IBM MQ Administrative Demo – https://youtu.be/cmhA12GwH1A

**RELIABILITY AND FAILOVER**

The following sections examine the high availability and reliability functionality of Apache ActiveMQ and IBM MQ, in case of issues related to network failure, server failure, and maintenance requirements.

**FAILURE SCENARIOS**

To test the reliability of the respective vendor solutions, a number of tests were completed to simulate common (and some not so common) events that can happen in the data center that might cause loss or duplication of messages. Since the focus of this report is on the high availability provided by the messaging solutions, it is assumed that the network attached storage and databases used for persistent storage are already highly available and will not be considered in these tests. The following provides some details of the failure tests that were performed and how they were simulated in the test environment.

- **Temporary Network Interruption:** Likely one of the most common scenarios, timeouts can occur for a variety of reasons (e.g., a temporary lack of network connectivity). To test a temporary network failure, the virtual network connection is temporarily disabled through the underlying virtualization software.
while the virtual machine is running. Figure 14 provides a view of the network facilities available through the VirtualBox virtualization software used for testing.

Figure 14: Simulate Network Interruption

- **Regular Scheduled Maintenance**: Regular maintenance could include both hardware upgrades and software upgrades, and patch management. The difference is that the running services are shutdown in a predictable manner. For this scenario, the running brokers/queue managers on the selected server were stopped and the server is converted into maintenance mode in a predictable manner. This includes a proper shutdown and reboot.

- **Server Failure**: While servers can fail for a number of reasons, for testing purposes the assumption is that the selected server failure is due to power supply failure. Using commodity hardware (without redundant power supplies), failure of the power supply provides an abrupt catastrophic failure that will not expectedly be recovered from quickly. To test a complete server failure in the test environment, a running virtual machine is forcibly powered off without a proper shutdown procedure. Figure 15 provides a visual example of this procedure through the VirtualBox virtualization software.
Figure 15: Simulate Server Failure

- **Multi-Server Failure:** The virtualized environments have been separated between two physical machines to simulate physical server placement on two separate racks within the same datacenter (note: geographically dispersed disaster recovery was not simulated). This is intended to maintain high availability in the event of a rack switch failure. To simulate this failure, the physical network cable can be disconnected from one of the servers interrupting communication with multiple servers simultaneously.

Each test simulation listed above was used for testing both the Point-to-Point (using both Fire-and-Forget and Request-Reply) and Publish-and-Subscribe models. For each of these tests, all servers were started in a known working state with all services running correctly to avoid compounding issues that might otherwise have cascading effects falsifying the results. Messages and responses were analyzed for any instances of duplicated or missing messages.

**SCENARIO 1: TEMPORARY NETWORK INTERRUPTION**

When using the default KahaDB on NFSv4 as the Apache ActiveMQ persistence store, an issue was encountered where messages can be duplicated and potentially lost. More specifically, when an Apache ActiveMQ Master/Slave setup loses network connectivity, the Slave correctly elevates to the Master role. However, the original Master comes back as a Master (as opposed to a Slave) when connectivity is re-established, operating with its own queue and functionality.

**POTENTIAL MESSAGE LOSS:** When a temporary network outage was introduced to Apache ActiveMQ, the Master broker incorrectly returned as a Master when the network connection was restored, despite the fact that its corresponding Slave had been elevated to the Master position. This meant that the originally recovered messages were still in the original Master’s queue, effectively duplicating the messages. While there are mechanisms that can be used to filter out duplicate messages, the larger concern is that the original Master and elevated Master continued to function independently with their own queue and functionality, without any corresponding Slave for failover. This means that should either one of these brokers fail with messages in the queue, the messages would be lost.
Designated as a “Split Brain” issue, there are existing tickets within the Apache JIRA system that refer to this issue, which have been identified and theoretically patched/fixed for the next release which currently has no expected release date at the time of writing.

In an attempt to eliminate the Apache ActiveMQ Split Brain issue, the “more reliable” JDBC was configured as the persistence store as opposed to the KahaDB on NFSv4. However, in this scenario, there was an issue with the original Master hanging. This meant that the original Master’s service port and admin console port were still open, which means the client will still attempt to connect to the service port and wait until timeout before moving to the next available broker. According to an Apache ActiveMQ JIRA ticket, the issue is not with the broker shutting down, but rather that the Java process was not shutting down because other items (e.g., Jetty used for the Web Console) may not be shut down so the Java process does not terminate. This means that when using JDBC as the persistent store, the administrative console must be removed from the broker configuration.

From the IBM perspective, when IBM MQ lost its network connection (and as a consequence its NFS access that stores all the information about the Queue Manager) it is “preemptively” stopped to prevent a situation of multiple active queue managers in a multi-instance setup. Ultimately, IBM MQ performed as expected without any issues.

For a detailed view into the Apache ActiveMQ Split Brain issue, and some insight into how the failover testing was performed, corresponding videos are available online at:

- Apache ActiveMQ High Availability Demo – https://youtu.be/dYJod-dPlIk
- IBM MQ High Availability Demo – https://youtu.be/iiLxRx9dY9w

**SCENARIO 2: SERVER MAINTENANCE**

Overall, both messaging solutions worked as expected for the maintenance related (i.e., predictable shutdown) scenario, with no messages lost or duplicated. The action of restarting the ActiveMQ broker essentially means that the Split-Brain issue does not surface.

With respect to time to failover, when manually shutting down or stopping the Apache ActiveMQ broker or IBM MQ Queue Manager, the corresponding Slave/Standby provided failover in 10 to 20 seconds.

**SCENARIO 3: SERVER FAILURE**

During this test scenario, both messaging solutions worked as expected with no message loss or duplication; although failover times took longer than scenario 2 with its predictable shutdown. For example, the default (i.e., delay between lock attempts is configurable) for Apache ActiveMQ typically required 90 seconds while IBM MQ was typically around 60 seconds. These defaults are configurable depending on individual organizational requirements.

**SCENARIO 4: MULTIPLE SERVER FAILURE**

In the tested environment, the results of the multiple server failure were exactly the same as the single version (i.e., Scenario 1). More specifically, IBM MQ worked as expected while Apache ActiveMQ essentially saw the already discussed Split Brain issue. The only difference here was the fact that the issue was duplicated given that it was multiple servers that were disconnected.
**FINAL ANALYSIS**

With the exception of the Split Brain issue associated with Apache ActiveMQ when there is a temporary network failure, both solutions can work as expected in different failure situations. More specifically, Apache ActiveMQ works as expected when the broker is stopped, either manually for maintenance or due to an unexpected power failure. But in the case of a temporary network interruption ActiveMQ runs the risk of message loss and/or duplicated messages, whether using KahaDB or JDBC as the persistent message store. This scenario alone instills enough concern to question the use of Apache ActiveMQ in an environment where message reliability is required.

<table>
<thead>
<tr>
<th>Task</th>
<th>Apache ActiveMQ</th>
<th>IBM MQ</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Temporary Network Interruption</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Point-to-Point: Fire and Forget</td>
<td>Failed</td>
<td>Passed</td>
</tr>
<tr>
<td>Point-to-Point: Request-Reply</td>
<td>Failed</td>
<td>Passed</td>
</tr>
<tr>
<td>Publish-and-Subscribe</td>
<td>Failed</td>
<td>Passed</td>
</tr>
<tr>
<td><strong>Temporary Network Interruption</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Point-to-Point: Fire and Forget</td>
<td>Passed</td>
<td>Passed</td>
</tr>
<tr>
<td>Point-to-Point: Request-Reply</td>
<td>Passed</td>
<td>Passed</td>
</tr>
<tr>
<td>Publish-and-Subscribe</td>
<td>Passed</td>
<td>Passed</td>
</tr>
<tr>
<td><strong>Temporary Network Interruption</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Point-to-Point: Fire and Forget</td>
<td>Passed</td>
<td>Passed</td>
</tr>
<tr>
<td>Point-to-Point: Request-Reply</td>
<td>Passed</td>
<td>Passed</td>
</tr>
<tr>
<td>Publish-and-Subscribe</td>
<td>Passed</td>
<td>Passed</td>
</tr>
<tr>
<td><strong>Multi-Server Failure</strong></td>
<td>Failed</td>
<td>Passed</td>
</tr>
</tbody>
</table>

As a reminder, Apache ActiveMQ Apollo is a recent development based on the experience of the ActiveMQ Project. In order to be “faster, more reliable, and easier to maintain” than ActiveMQ, a completely new architecture is being introduced, which may have a significant impact on the future development of the existing ActiveMQ incarnation. Organizations looking at implementing a messaging solution today, may need to keep this in mind for fear they may need to migrate between solutions in the near future.

In situations such as small implementations or the development of an application proof of concept, Apache ActiveMQ may provide a simpler setup. However, message reliability, implementations expected to see significant growth, aspects of maintenance and long-term management of the environment need to be considered. Once messaging environments reach a certain size their related maintenance costs may become cost prohibitive.

That being said, the recommendation is not to start with one and switch once that cost tipping point is reached; which will be different for each organization. The migration costs alone may prove cost prohibitive. Based on long-term goals and the anticipated increase in messaging requirements, starting with a commercial solution may prove more cost effective.

From the administrative perspective, IBM MQ provides more functionality and granular control over objects in the Queue Manager, while Apache ActiveMQ comparatively provides basic controls and functionality;
often requiring XML configuration modification and broker restarts. While this means that IBM MQ can sometimes appear more complicated, IBM provides functionality that helps eliminate much of the configuration complexity. For example, IBM MQ does not require manually editing configuration files and IBM MQ Explorer provides significant ease of use without a requirement to know all the command line switches and configuration options. Further, IBM MQ enables administrators to perform virtually all actions through its various interfaces compared to Apache ActiveMQ, which splits tasks between its interfaces forcing administrators to switch between them.

Finally, IBM MQ Explorer provides the facilities to manage the entire IBM MQ environment through a single window interface (including Standby Queue Managers), while IBM MQ’s support for scripting enables the automation of administrative and management tasks. Particularly in larger implementations, IBM MQ ultimately provides more facilities to ease management of the messaging environment compared to Apache ActiveMQ.
COMMISSIONED RESEARCH

IBM commissioned Zibis Group to perform an objective functional comparison of IBM MQ 8.0 and Apache ActiveMQ 5.11.1. The study took place between April 2015 and June 2015, with the analysis consisting of running a series of tests to assess how each solution performs from the perspective of high availability, failover, and administration. Zibis Group wholly supports the integrity and the methodology of how this study was conducted as well as the accuracy of the results and the conclusions. Zibis Group has made every attempt to provide a fair, honest, and unbiased research study of these messaging solutions through a set of defined Proof of Concepts.

ABOUT ZIBIS GROUP

Zibis Group Inc. (“Zibis Group”) is a leading source for market research and advisory services on the Information and Communications Technology (ICT) industry and related sectors. We assist vendors, government institutions, academia, non-profits and other key industry stakeholders by providing them with the tools they need to make informed decisions.

From those in product management, product marketing, and sales, to those in an economic development capacity, government agency or IT department (buyers), Zibis Group provides the needed insight and perspective as part of the decision making process to help advance the corporate agenda.

With over 15 years of experience, our team has the knowledge and ability to execute. Our focus is on working with the client, through a relationship-based model, allowing them to manage risk more effectively, and enable better business outcomes.

Zibis Group is a niche play in the marketplace for customized ICT sector research. We understand that all companies have distinct interests and unique business challenges. We thrive on rising to the occasion and in seeing our clients succeed. Technology is changing and we’re here to help.

For more information regarding Zibis Group, please visit http://zibisgroup.com.

DISCLAIMER

The views, analysis and opinion expressed in this document are those of Zibis Group Inc. and are based upon information obtained from public sources that are believed to be reliable. No independent verification has been made as to accuracy and completeness of such publicly available information. To the best of Zibis Group’s knowledge and belief, the information contained in this document is correct as on the date of this document. The opinion expressed in this document is Zibis Group’s current opinion and is subject to change without notice.

This documentation is provided “as is” without any warranty whatsoever and to the maximum extent permitted, Zibis Group disclaims all implied warranties, including without limitation the implied warranties of merchantability, non infringement and fitness for a particular purpose, with respect to the same. Zibis Group shall have no liability for errors, omissions or inadequacies in the information contained herein or for interpretations thereof. Zibis Group shall not be responsible for any damages, including without limitation, direct, indirect, consequential or incidental damages, arising out of the use of, or otherwise related to, this documentation or any other documentation. Nothing in this documentation is intended to be a recommendation of a specific product or company or intended to constitute an offer to buy or sell, or the solicitation of an offer to buy or sell, any product.

Reproduction and/or distribution of this document in any form without the prior written authorization of Zibis Group Inc. is strictly prohibited.

Copyright © 2015 Zibis Group Inc. Zibis Group and the Zibis Group logo are trademarks or registered trademarks of Zibis Group Inc. in Canada and other countries. Other product and company names mentioned herein may be trademarks of their respective owners.