Red Hat A-MQ 7.0 JMS Client

Installing A-MQ 7.0 Interconnect

Red Hat Customer Content Services

Red Hat A-MQ 7.0 JMS Client

Installing A-MQ 7.0 Interconnect

Legal Notice

Abstract

This guide provides guidance on installing A-MQ 7.0 Interconnect.

Table of Contents

CHAPTER 1. INSTALLING A-MQ 7.0 INTERCONNECT

1.1. INSTALLING JMS CLIENT (RPM INSTALLATION)

1.1.1. Installing OpenJDK on Red Hat Linux

You need to install OpenJDK to run JMS Client which is written in Java.

1. Subscribe to the Base Channel Obtain the OpenJDK from the RHN base channel. (Your installation of Red Hat Enterprise Linux is subscribed to this channel by default.)
2. To install the OpenJDK, run:

```
sudo yum install java-1.8.0-openjdk-devel
```

3. Verify that OpenJDK is now your system default. You can ensure the correct OpenJDK is set as the system default by following the steps below.

    a. As the root user, run：

    ```
    /usr/sbin/alternatives --config java
    ```

    b. Select **`/usr/lib/jvm /jre-1.8.0-openjdk/bin/java`**.

    c. As the root user, run：

    ```
     /usr/sbin/alternatives --config javac
    ```

    d. Select **`/usr/lib/jvm /java-1.8.0-openjdk/bin/javac`**.

1.1.2. Downloading JMS Client RPM Packages

The JMS Client RPM Packages for RHEL 6 are available here.

The JMS Client RPM Packages for RHEL 7 are available here.

You should download the following packages

- qpid-jms-client
- qpid-jms-client
- qpid-jms-client-examples
- qpid-jms-client-maven-repo

1.1.3. Installing JMS Client

As the root user, use the following command to install JMS Client:

```
yum install qpid-cpp-client qpid-cpp-client-devel
```

Note

The above instructions are only for the Alpha release.

1.1.4. Required Subscriptions

To install the JMS Client packages on Red Hat Enterprise Linux (RHEL), you require subscriptions to both of the following Red Hat products:

- Red Hat JBoss A-MQ 7
- RHEL 6 or 7

1.1.5. RPM Repositories

Enable the requisite RPM repository for your version of RHEL, as shown in the following table:

Table 1.1. RPM Repositories for JMS Clients

Platform Version	RPM Repository
RHEL 7	a-mq-clients-1-for-rhel-7-server-rpms
RHEL 6	a-mq-clients-1-for-rhel-6-server-rpms

1.1.6. Registering Your System

To install the JMS Client installer, first register the host system using Red Hat Subscription Manager, and subscribe to the required channels.

1. Register your system with the Content Delivery Network, entering your Customer Portal user name and password when prompted.

   ```
   sudo subscription-manager register
   ```

2. Find the entitlement pool for the Red Hat JMS 7 Clients director.

   ```
   sudo subscription-manager list --available --all
   ```

3. Use the pool ID located in the previous step to attach the JMS Clients entitlements for corresponding platforms.

   ```
   sudo subscription-manager attach --pool=<POOL_ID>
   ```

4. Disable all default repositories then enable the required Red Hat Enterprise Linux repositories.

```
sudo subscription-manager repos --disable=*
sudo subscription-manager repos --enable=a-mq-clients-1-for-rhel-
7-server-rpms --enable=a-mq-clients-1-for-rhel-6-server-rpms --
enable=a-mq-clients-1-for-rhel-5-server-rpms
```

5. Perform an update on your system to make sure you have the latest base system packages.

```
sudo yum update -y
reboot
```

The system is now ready for the director installation.

1.1.7. AMQP JMS A-MQ 1.0 Java Client

Qpid JMS is a complete AMQP JMS 1.0 and JMS 1.1 client built using Qpid Proton.

To install the AMQP JMS A-MQ 1.0, enter the following command as the root user:

```
yum install qpid-jms-client qpid-jms-client-docs qpid-jms-client-
examples qpid-jms-client-maven-repo
```

Note

Both AMQP JMS 1.0 and JMS 1.1 clients require Java 1.7 or higher runtime to run. Ensure the Java version installed on your system is 1.7 or higher.

1.1.8. AMQP C++ Client

To install the AMQP C++ client, enter the following command as the root user:

```
yum install qpid-cpp-client qpid-cpp-client-devel
```

1.1.9. Python Language Bindings for AMQP API

To install the Python language bindings for the AMQP API messaging framework, enter the following command as the root user:

```
yum install python-qpid-proton python-qpid-proton-doc
```

1.2. INSTALLING JMS CLIENT (ZIP INSTALLATION)

The ZIP file installation is platform-independent.

1. Download the JMS Client RPM package from here.
2. Extract the package by using the following command:

```
rpm2cpio qpid-jms-client-zip-0.8.0-1.el6.noarch.rpm | cpio -ivd
```

You can find the JMS Client ZIP file (apache-qpid-jms-0.8.0.redhat-1-bin.zip) in the

usr/share/java/ directory.

3. Extract the package contents into a directory to which you have full access.

Note

The above instructions are only for the Alpha release.

1.2.1. Downloading the A-MQ Broker ZIP File From the Red Hat Customer Portal

1. Open a browser and log in to the Red Hat Customer Portal at https://access.redhat.com.
2. Click **Downloads**.
3. Click **Red Hat JBoss Middleware** in the **Product Downloads** list.
4. Select the correct A-MQ version from the **Version** drop-down menu.
5. Find **Red Hat A-MQ JMS Client** in the list and click the **Download** link.

1.2.2. Unpacking the JMS Client ZIP File

Once you have downloaded the JMS Client ZIP installation file, you can install it by extracting the package contents into a directory to which you have full access.

1.3. JMS CLIENT CONFIGURATION

This section details various configuration options for the client, such as how to configure and create a JNDI InitialContext, the syntax for its related configuration, and various URI options that can be set when defining a ConnectionFactory.

1.3.1. Configuring A JNDI InitialContext

Applications use a **JNDI InitialContext**, itself obtained from an **InitialContextFactory**, to look up JMS objects such as **ConnectionFactory**. The JMS client provides an implementation of the **InitialContextFactory** in class **org.apache.qpid.jms.jndi.JmsInitialContextFactory**. This can be configured and used in three main ways:

1. Via the **jndi.properties** file on the Java Classpath.

By including a file named **jndi.properties** on the Classpath and setting the **java.naming.factory.initial** property to **org.apache.qpid.jms.jndi.JmsInitialContextFactory**, the Qpid InitialContextFactory implementation will be discovered when instantiating InitialContext object.

```
javax.naming.Context ctx = new javax.naming.InitialContext();
```

2. Via system properties.

By setting the **`java.naming.factory.initial`** system property to value **`org.apache.qpid.jms.jndi.JmsInitialContextFactory`**, the Qpid InitialContextFactory implementation will be discovered when instantiating InitialContext object.

```
javax.naming.Context ctx = new javax.naming.InitialContext();
```

The particular ConnectionFactory, Queue and Topic objects you wish the context to contain are configured as properties in a file, which is passed using the **`java.naming.provider.url`** system property. The syntax for these properties is detailed below.

3. Programmatically using an environment **`Hashtable`**.

The InitialContext can also be configured directly by passing an environment during creation:

```
Hashtable<Object, Object> env = new Hashtable<Object, Object>();
env.put(Context.INITIAL_CONTEXT_FACTORY,
"org.apache.qpid.jms.jndi.JmsInitialContextFactory");
javax.naming.Context context = new javax.naming.InitialContext(env);
```

The particular ConnectionFactory, Queue and Topic objects you wish the context to contain are configured as properties (the syntax for which is detailed below), either directly within the environment **`Hashtable`**, or in a separate file which is referenced using the **`java.naming.provider.url`** property within the environment **`Hashtable`**.

The property syntax used in the properties file or environment Hashtable is as follows: - To define a ConnectionFactory, use format: **`connectionfactory.lookupName = URI`** - To define a Queue, use format: **`queue.lookupName = queueName`** - To define a Topic use format: **`topic.lookupName = topicName`**

As an example, consider the following properties used to define a ConnectionFactory, Queue, and Topic:

```
connectionfactory.myFactoryLookup = amqp://localhost:5672
queue.myQueueLookup = queueA
topic.myTopicLookup = topicA
```

These objects can then be looked up from a Context as follows:

```
ConnectionFactory factory = (ConnectionFactory)
context.lookup("myFactoryLookup");
Queue queue = (Queue) context.lookup("myQueueLookup");
Topic topic = (Topic) context.lookup("myTopicLookup");
```

1.3.2. Connection URI

The basic format of the clients Connection URI is:

```
amqp://hostname:port[?option=value[&option2=value...]]
```

The client can be configured with a number of different settings using the URI while defining the ConnectionFactory, these are detailed in the following sections.

1.3.2.1. JMS Configuration Options

The options apply to the behaviour of the JMS objects such as Connection, Session, MessageConsumer and MessageProducer.

Table 1.2. JMS Configuration options

Parameter	Description
jms.username	User name value used to authenticate the connection
jms.password	The password value used to authenticate the connection
jms.clientID	The ClientID value that is applied to the connection
jms.forceAsyncSend	Configures whether all Messages sent from a MessageProducer are sent asynchronously or only those Message that qualify such as Messages inside a transaction or non-persistent messages
jms.forceSyncSend	Override all asynchronous send conditions and always sends every Message from a MessageProducer synchronously
jms.forceAsyncAcks	Causes all Message acknowledgments to be sent asynchronously
jms.alwaysSyncSend	Override all asynchronous send conditions and always sends every Message from a MessageProducer synchronously.
jms.sendAcksAsync	Causes all Message acknowledgments to be sent asynchronously.

jms.localMessageExpiry	Controls whether MessageConsumer instances will locally filter expired Messages or deliver them. By default this value is set to true and expired messages will be filtered
jms.localMessagePriority	If enabled prefetched messages are reordered locally based on their given Message priority value. Default is false.
jms.validatePropertyNames	If message property names should be validated as valid Java identifiers. Default is true.
jms.receiveLocalOnly	If enabled receive calls with a timeout will only check a consumers local message buffer, otherwise the remote peer is checked to ensure there are really no messages available if the local timeout expires before a message arrives. Default is false, the remote is checked.
jms.receiveNoWaitLocalOnly	If enabled receiveNoWait calls will only check a consumers local message buffer, otherwise the remote peer is checked to ensure there are really no messages available. Default is false, the remote is checked.
jms.queuePrefix	Optional prefix value added to the name of any Queue created from a JMS Session.
jms.topicPrefix	Optional prefix value added to the name of any Topic created from a JMS Session.
jms.closeTimeout	Timeout value that controls how long the client waits on Connection close before returning. (By default the client waits 15 seconds for a normal close completion event).
jms.connectTimeout	Timeout value that controls how long the client waits on Connection establishment before returning with an error. (By default the client waits 15 seconds for a connection to be established before failing).

jms.clientIDPrefix	Optional prefix value that is used for generated Client ID values when a new Connection is created for the JMS ConnectionFactory. The default prefix is *ID:*.
jms.connectionIDPrefix	Optional prefix value that is used for generated Connection ID values when a new Connection is created for the JMS ConnectionFactory. This connection ID is used when logging some information from the JMS Connection object so a configurable prefix can make breadcrumbing the logs easier. The default prefix is *ID:*.
jms.messageIDType	Controls the type of the Message ID assigned to messages sent from the client. By default a generated String value is used on outgoing messages, other available types are UUID and UUID_STRING.

These values control how many messages the remote peer can send to the client and be held in a prefetch buffer for each consumer instance.

Parameter	**Default Value**
jms.prefetchPolicy.queuePrefetch	1000
jms.prefetchPolicy.topicPrefetch	1000
jms.prefetchPolicy.queueBrowserPrefetch	1000
jms.prefetchPolicy.durableTopicPrefetch	1000

jms.prefetchPolicy.all	Used to set all prefetch values at once. The RedeliveryPolicy controls how redelivered messages are handled on the client.
jms.redeliveryPolicy.maxRedeliveries	Controls when an incoming message is rejected based on the number of times it has been redelivered, the default value is (-1) disabled. A value of zero would indicate no message redeliveries are accepted, a value of five would allow a message to be redelivered five times, etc.

1.3.2.2. TCP Transport Configuration Options

When connected to a remote using plain TCP these options configure the behaviour of the underlying socket. These options are appended to the connection URI along with the other configuration options, for example:

```
amqp://localhost:5672?jms.clientID=foo&transport.connectTimeout=30000
```

The complete set of TCP Transport options is listed below:

Parameter	**Default Value**
transport.sendBufferSize	64k
transport.receiveBufferSize	64k
transport.trafficClass	0
transport.connectTimeout	60 seconds
transport.soTimeout	-1
transport.soLinger	-1
transport.tcpKeepAlive	False

transport.tcpNoDelay	True

1.3.2.3. SSL Transport Configuration Options

The SSL Transport extends the TCP Transport and is enabled using the amqps URI scheme. Because the SSL Transport extends the functionality of the TCP based Transport all the TCP Transport options are valid on an SSL Transport URI.

A simple SSL based client URI is shown below:

```
amqps://localhost:5673
```

The complete set of SSL Transport options is listed below:

Parameter	Description
transport.keyStoreLocation	Default is to read from the system property "javax.net.ssl.keyStore"
transport.keyStorePassword	Default is to read from the system property "javax.net.ssl.keyStorePassword"
transport.trustStoreLocation	Default is to read from the system property "javax.net.ssl.trustStore"
transport.trustStorePassword	Default is to read from the system property "javax.net.ssl.keyStorePassword"
transport.storeType	The type of trust store being used. Default is "JKS".
transport.contextProtocol	The protocol argument used when getting an SSLContext. Default is "TLS".
transport.enabledCipherSuites	The cipher suites to enable, comma separated. No default, meaning the context default ciphers are used. Any disabled ciphers are removed from this.

transport.disabledCipherSuites	The cipher suites to disable, comma separated. Ciphers listed here are removed from the enabled ciphers. No default.
transport.enabledProtocols	The protocols to enable, comma separated. No default, meaning the context default protocols are used. Any disabled protocols are removed from this.
transport.disabledProtocols	The protocols to disable, comma separated. Protocols listed here are removed from the enabled protocols. Default is "SSLv2Hello,SSLv3".
transport.trustAll	Whether to trust the provided server certificate implicitly, regardless of any configured trust store. Defaults to false.
transport.verifyHost	Whether to verify that the hostname being connected to matches with the provided server certificate. Defaults to true.
transport.keyAlias	The alias to use when selecting a keypair from the keystore if required to send a client certificate to the server. No default.

1.3.2.4. AMQP Configuration Options

These options apply to the behaviour of certain AMQP functionality.

Parameter	Description
amqp.idleTimeout	The idle timeout in milliseconds after which the connection will be failed if the peer sends no AMQP frames. Default is 60000.
amqp.vhost	The vhost to connect to. Used to populate the Sasl and Open hostname fields. Default is the main hostname from the Connection URI.

Parameter	Description
amqp.saslLayer	Controls whether connections should use a SASL layer or not. Default is true.
amqp.saslMechanisms	Which SASL mechanism(s) the client should allow selection of, if offered by the server and usable with the configured credentials. Comma separated if specifying more than 1 mechanism. Default is to allow selection from all the clients supported mechanisms, which are currently EXTERNAL, CRAM-MD5, PLAIN, and ANONYMOUS.
amqp.maxFrameSize	The max-frame-size value in bytes that is advertised to the peer. Default is 1048576.

1.3.2.5. Failover Configuration Options

With failover enabled the client can reconnect to a different broker automatically when the connection to the current connection is lost for some reason. The failover URI is always initiated with the failover prefix and a list of URIs for the brokers is contained inside a set of parentheses. The "jms." options are applied to the overall failover URI, outside the parentheses, and affect the JMS Connection object for its lifetime.

The URI for failover looks something like the following:

```
failover:(amqp://host1:5672,amqp://host2:5672)?
jms.clientID=foo&failover.maxReconnectAttempts=20
```

The individual broker details within the parentheses can use the "transport." or "amqp." options defined earlier, with these being applied as each host is connected to:

```
failover:(amqp://host1:5672?amqp.option=value,amqp://host2:5672?
transport.option=value)?jms.clientID=foo
```

The complete set of configuration options for failover is listed below:

Parameter	Description
failover.initialReconnectDelay	The amount of time the client will wait before the first attempt to reconnect to a remote peer. The default value is zero, meaning the first attempt happens immediately.

Parameter	Description
failover.reconnect	Delay Controls the delay between successive reconnection attempts, defaults to 10 milliseconds. If the backoff option is not enabled this value remains constant.
failover.maxReconnect	Delay The maximum time that the client will wait before attempting a reconnect. This value is only used when the backoff feature is enabled to ensure that the delay doesn't not grow too large. Defaults to 30 seconds as the max time between connect attempts.
failover.useReconnectBackOff	Controls whether the time between reconnection attempts should grow based on a configured multiplier. This option defaults to true.
failover.reconnectBackOffMultiplier	The multiplier used to grow the reconnection delay value, defaults to 2.0d.
failover.maxReconnectAttempts	The number of reconnection attempts allowed before reporting the connection as failed to the client. The default is no limit or (-1).
failover.startupMaxReconnectAttempts	For a client that has never connected to a remote peer before this option control how many attempts are made to connect before reporting the connection as failed. The default is to use the value of maxReconnectAttempts.
failover.warnAfterReconnectAttempts	Controls how often the client will log a message indicating that failover reconnection is being attempted. The default is to log every 10 connection attempts.

The failover URI also supports defining *nested* options as a means of specifying AMQP and transport option values applicable to all the individual nested broker URI's, which can be useful to avoid repetition. This is accomplished using the same "transport." and "amqp." URI options outlined earlier for a non-failover broker URI but prefixed with failover.nested.. For example, to apply the same value for the amqp.vhost option to every broker connected to you might have a URI like:

```
failover:(amqp://host1:5672,amqp://host2:5672)?
jms.clientID=foo&failover.nested.amqp.vhost=myhost
```

1.3.2.6. Discovery Configuration Options

The client has an optional Discovery module, which provides a customized failover layer where the broker URIs to connect to are not given in the initial URI, but discovered as the client operates via associated discovery agents. There are currently two discovery agent implementations, a file watcher that loads URIs from a file, and a multicast listener that works with ActiveMQ 5 brokers which have been configured to broadcast their broker addresses for listening clients.

The general set of failover related options when using discovery are the same as those detailed earlier, with the main prefix updated from failover. to discovery., and with the *nested* options prefix used to supply URI options common to all the discovered broker URIs bring updated from failover.nested. to discovery.discovered. For example, without the agent URI details, a general discovery URI might look like:

```
discovery:(<agent-uri>)?
discovery.maxReconnectAttempts=20&discovery.discovered.jms.clientID=foo
```

To use the file watcher discovery agent, utilize an agent URI of the form:

```
discovery:(file:///path/to/monitored-file?updateInterval=60000)
```

The URI options for the file watcher discovery agent are listed below:

- **updateInterval** Controls the frequency in milliseconds which the file is inspected for change. The default value is 30000.

To use the multicast discovery agent with an ActiveMQ 5 broker, utilize an agent URI of the form:

```
discovery:(multicast://default?group=default)
```

Note that the use of default as the host in the multicast agent URI above is a special value (that is substituted by the agent with the default "239.255.2.3:6155"). You can change this to specify the actual IP and port in use with your multicast configuration.

- **group** Controls which multicast group messages are listened for on. The default value is "default".

1.3.3. Logging

The client makes use of the SLF4J API, allowing users to select a particular logging implementation based on their needs by supplying a SLF4J *binding*, such as slf4j-log4j in order to use Log4J. More details on SLF4J are available from http://www.slf4j.org/.

The client uses Logger names residing within the org.apache.qpid.jms hierarchy, which you can use to configure a logging implementation based on your needs.

When debugging some issues, it can be useful to enable additional protocol trace logging from the Qpid Proton AMQP 1.0 library. There are two options to achieve this:

- Set the environment variable (not Java system property) **`PN_TRACE_FRM`** to true, which will cause Proton to emit frame logging to stdout.
- Add the option **`amqp.traceFrames=true`** to your connection URI to have the client add a protocol tracer to Proton, and configure the **`org.apache.qpid.jms.provider.amqp.FRAMES Logger`** to **`TRACE`** level to include the output in your logs.