AMQP Messaging Broker (Java)

AMQP Messaging Broker (Java)

Table of Contents

List of Figures

List of Tables

List of Examples

Chapter 1. Introduction

The Java Broker is a powerful open-source message broker that implements all versions of the Advanced Message Queuing Protocol (AMQP) [http://www.amqp.org]. The Java Broker is actually one of two message brokers provided by the Apache Qpid project [http://qpid.apache.org]: the Java Broker and the C++ Broker.

This document relates to the Java Broker. The C++ Broker is described separately [../../AMQP-Messaging-Broker-CPP-Book/html/].

Headline features

- 100% Java implementation - runs on any platform supporting Java 1.6 or higher
- Messaging clients support in Java, C++, Python.
- JMS 1.1 compliance (Java client).
- Persistent and non-persistent (transient) message support
- Supports for all common messaging patterns (point-to-point, publish/subscribe, fan-out etc).
- Transaction support including XA[1]
- Supports for all versions of the AMQP protocol
- Automatic message translation, allowing clients using different AMQP versions to communicate with each other.
- Pluggable authentication architecture with out-of-the-box support for Kerberos, LDAP, External, and file-based authentication mechanisms.
- Pluggable message store architecture with implementations based on Apache Derby [http://db.apache.org/derby/], Oracle BDB JE [http://www.oracle.com/technetwork/products/berkeleydb/overview/index-093405.html][2], and Memory Store
- Web based management interface and programmatic management interfaces via REST and JMX APIs.
- SSL support
- High availability (HA) support.[3]

[1] XA provided when using AMQP 0-10
[2] Oracle BDB JE must be downloaded separately.
[3] HA currently only available to users of the optional BDB JE HA based message store.

Chapter 2. Installation

2.1. Introduction

This document describes how to install the Java Broker on both Windows and UNIX platforms.

2.2. Prerequisites

2.2.1. Java Platform

The Java Broker is an 100% Java implementation and as such it can be used on any operating system supporting Java 1.6 or higher. This includes Linux, Solaris, Mac OS X, and Windows XP/Vista/7/8.

The broker has been tested with Java implementations from both Oracle and IBM. Whatever platform you chose, it is recommended that you ensure it is patched with any critical updates made available from the vendor.

Verify that your JVM is installed properly by following these instructions.

2.2.2. Disk

The Java Broker installation requires approximately 20MB of free disk space.

The Java Broker also requires a working directory. The working directory is used for the message store, that is, the area of the file-system used to record persistent messages whilst they are passing through the Broker. The working directory is also used for the default location of the log file. The size of the working directory will depend on the how the Broker is used.

The performance of the file system hosting the work directory is key to the performance of Broker as a whole. For best performance, choose a device that has low latency and one that is uncontended by other applications.

Be aware that there are additional considerations if you are considering hosting the working directory on NFS. See Chapter 8, *Stores* for further details.

2.2.3. Memory

Qpid caches messages on the heap for performance reasons, so in general, the Broker will benefit from as much heap as possible. However, on a 32bit JVM, the maximum addressable memory range for a process is 4GB, after leaving space for the JVM's own use this will give a maximum heap size of approximately ~3.7GB.

2.2.4. Operating System Account

Installation or operation of Qpid does *not* require a privileged account (i.e. root on UNIX platforms or Administrator on Windows). However it is suggested that you use an dedicated account (e.g. qpid) for the installation and operation of the Java Broker.

2.3. Download

2.3.1. Broker Release

You can download the latest qpid-java-broker-0.22.tar.gz package from the Download Page [http://qpid.apache.org/download.html].

It is recommended that you confirm the integrity of the download by verifying the PGP signature matches that available on the site. Instrutions are given on the download page.

2.3.2. Optional Dependencies

The broker has an optional message store implementations backed by Oracle BDB JE. If you wish to use these stores you will need to provide the optional Oracle BDB JE dependency. For more details, see Section 8.4, “BDB Store”

2.4. Installation on Windows

Firstly, verify that your JVM is installed properly by following these instructions.

Now chose a directory for Qpid broker installation. This directory will be used for the Qpid JARs and configuration files. It need not be the same location as the store used for the persistent messages or the log file (you will chose this location later). For the remainder this example we will assumed that location c:\qpid has been chosen.

Now using WinZip[1] (or similar) extract the Qpid package qpid-java-broker-0.22.tar.gz into the directory.

The extraction of the Qpid package will have created a directory qpid-broker-0.22 within c:\qpid

```
Volume in drive C has no label

 Directory of c:\qpid\qpid-broker-0.22

07/25/2012  11:22 PM                 .
09/30/2012  10:51 AM                 ..
09/30/2012  12:24 AM                 bin
08/21/2012  11:17 PM                 etc
07/25/2012  11:22 PM                 lib
07/20/2012  08:10 PM            65,925 LICENSE
07/20/2012  08:10 PM             3,858 NOTICE
07/20/2012  08:10 PM             1,346 README.txt
               3 File(s)         71,129 bytes
               5 Dir(s)  743,228,796,928 bytes free
```

2.4.1. Setting the working directory

Qpid requires a work directory. This directory is used for the default location of the Qpid log file and is used for the storage of persistent messages. The work directory can be set on the command-line (for the lifetime of the command interpreter), but you will normally want to set the environment variable permanently via the Advanced System Settings in the Control Panel.

[1] WinZip is a Registered Trademark of WinZip International LLC

```
set QPID_WORK=C:\qpidwork
```

If the directory referred to by QPID_WORK does not exist, the Java Broker will attempt to create it on start-up.

2.4.2. Optional Dependencies

The broker has optional message store implementations backed by Oracle BDB JE. If you wish to use these stores you will need to provide the optional Oracle BDB JE dependency. For more details, see Section 8.4, “BDB Store”

2.5. Installation on UNIX platforms

Firstly, verify that your JVM is installed properly by following these instructions.

Now chose a directory for Qpid broker installation. This directory will be used for the Qpid JARs and configuration files. It need not be the same location as the store used for the persistent messages or the log file (you will chose this location later). For the remainder this example we will assumed that location /usr/local/qpid has been chosen.

Extract the Qpid package qpid-java-broker-0.22.tar.gz into the directory.

```
mkdir /usr/local/qpid
cd /usr/local/qpid
tar xvzf qpid-java-broker-0.22.tar.gz>
```

The extraction of the Qpid package will have created a directory qpid-broker-x.x

```
ls -la qpid-broker-0.22/
total 152
drwxr-xr-x   8 qpid  qpid    272 25 Jul 23:22 .
drwxr-xr-x  45 qpid  qpid   1530 30 Sep 10:51 ..
-rw-r--r--@  1 qpid  qpid  65925 20 Jul 20:10 LICENSE
-rw-r--r--@  1 qpid  qpid   3858 20 Jul 20:10 NOTICE
-rw-r--r--@  1 qpid  qpid   1346 20 Jul 20:10 README.txt
drwxr-xr-x  10 qpid  qpid    340 30 Sep 00:24 bin
drwxr-xr-x   9 qpid  qpid    306 21 Aug 23:17 etc
drwxr-xr-x  34 qpid  qpid   1156 25 Jul 23:22 lib
```

2.5.1. Setting the working directory

Qpid requires a work directory. This directory is used for the default location of the Qpid log file and is used for the storage of persistent messages. The work directory can be set on the command-line (for the lifetime of the current shell), but you will normally want to set the environment variable permanently the user's shell profile file (~/.bash_profile for Bash etc).

```
export QPID_WORK=/var/qpidwork
```

If the directory referred to by QPID_WORK does not exist, the Java Broker will attempt to create it on start-up.

2.5.2. Optional Dependencies

The broker has an optional message store implementations backed by Oracle BDB JE. If you wish to use these stores you will need to provide the optional Oracle BDB JE dependency. For more details, see Section 8.4, “BDB Store”

Chapter 3. Getting Started

This section describes how to start the Java Broker for the first time.

3.1. Starting/Stopping the Broker

To start the Broker, use the **qpid-server** script (UNIX) or **qpid-server.bat** (Windows) provided within distribution.

3.2. Starting/Stopping on Windows

Firstly change to the installation directory used during the installation and ensure that the QPID_WORK environment variable is set.

Now use the **qpid-server.bat** to start the server

```
bin\qpid-server.bat
```

Output similar to the following will be seen:

```
[Broker] BRK-1006 : Using configuration : C:\qpid\qpid-broker-0.22\etc\config.xml
[Broker] BRK-1007 : Using logging configuration : C:\qpid\qpid-broker-0.22\etc\log
[Broker] BRK-1001 : Startup : Version: 0.22 Build: 1411386
[Broker] BRK-1010 : Platform : JVM : Sun Microsystems Inc. version: 1.6.0_24-b07 O
[Broker] BRK-1011 : Maximum Memory : 1,069,416,448 bytes
[Broker] MNG-1001 : Web Management Startup
[Broker] MNG-1002 : Starting : HTTP : Listening on port 8080
[Broker] MNG-1004 : Web Management Ready
[Broker] MNG-1001 : JMX Management Startup
[Broker] MNG-1002 : Starting : RMI Registry : Listening on port 8999
[Broker] MNG-1002 : Starting : JMX RMIConnectorServer : Listening on port 9099
[Broker] MNG-1004 : JMX Management Ready
[Broker] BRK-1002 : Starting : Listening on TCP port 5672
[Broker] BRK-1004 : Qpid Broker Ready
```

The BRK-1004 message confirms that the Broker is ready for work. The MNG-1002 and BRK-1002 confirm the ports to which the Broker is listening (for HTTP/JMX management and AMQP respectively).

To stop the Broker, use Control-C or use the Shutdown MBean made from the Section 9.4, “JMX”

3.3. Starting/Stopping on Unix

Firstly change to the installation directory used during the installation and ensure that the QPID_WORK environment variable is set.

Now use the **qpid-server** script to start the server:

```
bin\qpid-server
```

Output similar to the following will be seen:

```
[Broker] BRK-1006 : Using configuration : /usr/local/qpid/qpid-broker-0.22/etc/con
```

```
[Broker] BRK-1007 : Using logging configuration : /usr/local/qpid/qpid-broker-0.22
[Broker] BRK-1001 : Startup : Version: 0.22 Build: 1411386
[Broker] BRK-1010 : Platform : JVM : Apple Inc. version: 1.6.0_35-b10-428-11M3811
[Broker] BRK-1011 : Maximum Memory : 1,070,399,488 bytes
[Broker] MNG-1001 : Web Management Startup
[Broker] MNG-1002 : Starting : HTTP : Listening on port 8080
[Broker] MNG-1004 : Web Management Ready
[Broker] MNG-1001 : JMX Management Startup
[Broker] MNG-1002 : Starting : RMI Registry : Listening on port 8999
[Broker] MNG-1002 : Starting : JMX RMIConnectorServer : Listening on port 9099
[Broker] MNG-1004 : JMX Management Ready
[Broker] BRK-1002 : Starting : Listening on TCP port 5672
[Broker] BRK-1004 : Qpid Broker Ready
```

The BRK-1004 message confirms that the Broker is ready for work. The MNG-1002 and BRK-1002 confirm the ports to which the Broker is listening (for HTTP/JMX management and AMQP respectively).

To stop the Broker, use Control-C from the controlling shell, use the **bin/qpid.stop** script, use **kill -TERM <pid>**, or the Shutdown MBean from Section 9.4, “JMX”

3.4. Log file

The Java Broker writes a log file to record both details of its normal operation and any exceptional conditions. By default the log file is written within the log subdirectory beneath the work directory - `$QPID_WORK/log/qpid.log` (UNIX) and `%QPID_WORK%\log\qpid.log` (Windows).

For details of how to control the logging, see Section 11.1, “Log Files”

3.5. Using the command line

The Java Broker understands a number of command line options which may be used to override the configuration.

To see usage information for all command line options, use the option `--help`

```
bin/qpid-server --help

usage: Qpid [-cic <path>] [-h] [-icp <path>] [-l <file>] [-mm] [-mmhttp <port>]
            [-mmjmx <port>] [-mmpass <password>] [-mmqv] [-mmrmi <port>] [-os]
            [-sp <path>] [-st <type>] [-v] [-w <period>]
 -cic <path>                                    create a copy of the initial confi
 --create-initial-config <path>                 file, either to an optionally spec
                                                file path, or as initial-config.js
                                                in the current directory

 -h,                                            print this message
 --help

 -icp  <path>                                   set the location of initial JSON c
 --initial-config-path <path>                   to use when creating/overwriting a
                                                broker configuration store

 -l <file>                                      use the specified log4j xml config
 --logconfig <file>                             file. By default looks for a file
```

```
                                                  etc/log4j.xml in the same director
                                                  the configuration file

-mm                                               start broker in management mode,
                                                  disabling the AMQP ports

-mmhttp <port>                                    override http management port in
--management-mode-http-port <port>                management mode

-mmjmx                                            override jmx connector port in
--management-mode-jmx-connector-port <port>       management mode

-mmpass  <password>                               Set the password for the managemen
--management-mode-password <password>             mode user mm_admin

-mmqv                                             make virtualhosts stay in the quie
--management-mode-quiesce-virtualhosts            state during management mode.

-mmrmi <port>                                     override jmx rmi registry port in
--management-mode-rmi-registry-port <port>        management mode

-os                                               overwrite the broker configuration
--overwrite-store                                 with the current initial configura

-sp <path>                                        use given configuration store loca
--store-path <path>

-st <type>                                        use given broker configuration sto
--store-type <type>

-v                                                print the version information and
--version

-w <period>                                       monitor the log file configuration
--logwatch <period>                               for changes. Units are seconds. Ze
                                                  means do not check for changes.
```

Chapter 4. Concepts

4.1. Virtual Hosts

4.2. Exchanges

4.3. Queues

4.4. Ports

4.5. Protocols

4.6. Authentication Providers

4.7. Other Services

Chapter 5. Virtual Hosts

Chapter 6. Exchanges

Chapter 7. Queues

7.1. Messaging Groups

7.2. Other Queue Types

7.2.1. Introduction

In addition to the standard queue type where messages are delivered in the same order that they were sent, the Java Broker supports four additional queue types which allows for alternative delivery behaviours. These are priority-queues, sorted-queues-, last-value-queues (LVQs), and grouped queues.

In the following sections, the semantics of each queue type is described, followed by a description of how instances of these queue can be created via configuration or programmatically.

The final section discusses the importance of using a low client pre-fetch with these queued.

7.2.2. Priority Queues

In a priority queue, messages on the queue are delivered in an order determined by the JMS priority message header [http://docs.oracle.com/javaee/6/api/javax/jms/Message.html#getJMSPriority()] within the message. By default Qpid supports the 10 priority levels mandated by JMS, with priority value 0 as the lowest priority and 9 as the highest.

It is possible to reduce the effective number of priorities if desired.

JMS defines the default message priority [http://docs.oracle.com/javaee/6/api/javax/jms/Message.html#DEFAULT_PRIORITY] as 4. Messages sent without a specified priority use this default.

7.2.3. Sorted Queues

Sorted queues allow the message delivery order to be determined by value of an arbitrary JMS message property [http://docs.oracle.com/javaee/6/api/javax/jms/Message.html#getStringProperty()]. Sort order is alpha-numeric and the property value must have a type java.lang.String.

Messages sent to a sorted queue without the specified JMS message property will be inserted into the 'last' position in the queue.

7.2.4. Last Value Queues (LVQ)

LVQs (or conflation queues) are special queues that automatically discard any message when a newer message arrives with the same key value. The key is specified by arbitrary JMS message property [http://docs.oracle.com/javaee/6/api/javax/jms/Message.html#getPropertyNames()].

An example of an LVQ might be where a queue represents prices on a stock exchange: when you first consume from the queue you get the latest quote for each stock, and then as new prices come in you are sent only these updates.

Like other queues, LVQs can either be browsed or consumed from. When browsing an individual subscriber does not remove the message from the queue when receiving it. This allows for many subscriptions to browse the same LVQ (i.e. you do not need to create and bind a separate LVQ for each subscriber who wishes to receive the contents of the LVQ).

Messages sent to an LVQ without the specified property will be delivered as normal and will never be "replaced".

7.2.5. Creating a Priority, Sorted or LVQ Queue

To create a priority, sorted or LVQ queue, it can be defined in the virtualhost configuration file, or the queue can be created programmtically from a client via AMQP (using an extension to JMS), or using JMX. These methods are described below.

Once a queue is created you cannot change its type (without deleting it and re-creating). Also note you cannot currently mix the natures of these queue types, for instance, you cannot define a queue which it both an LVQ and a priority-queue.

7.2.5.1. Using configuration

To create a priority, sorted or LVQ queue within configuration, add the appropriate xml to the virtualhost.xml configuration file within the `queues` element.

7.2.5.1.1. Priority

To defining a priority queue, add a <priority>true</priority> element. By default the queue will have 10 distinct priorities.

Example 7.1. Configuring a priority queue

```
<queue>
    <name>myqueue</name>
    <myqueue>
        <exchange>amq.direct</exchange>
        <priority>true</priority>
    </myqueue>
</queue>
```

If you require fewer priorities, it is possible to specify a `priorities` element (whose value is a integer value between 2 and 10 inclusive) which will give the queue that number of distinct priorities. When messages are sent to that queue, their effective priority will be calculated by partitioning the priority space. If the number of effective priorities is 2, then messages with priority 0-4 are treated the same as "lower priority" and messages with priority 5-9 are treated equivalently as "higher priority".

Example 7.2. Configuring a priority queue with fewer priorities

```
<queue>
    <name>myqueue</name>
    <myqueue>
        <exchange>amq.direct</exchange>
        <priority>true</priority>
        <priorities>4</priorities>
    </myqueue>
</queue>
```

7.2.5.1.2. Sorted

To define a sorted queue, add a `sortKey` element. The value of the `sortKey` element defines the message property to use the value of when sorting the messages put onto the queue.

Example 7.3. Configuring a sorted queue

```
<queue>
    <name>myqueue</name>
    <myqueue>
        <exchange>amq.direct</exchange>
        <sortKey>message-property-to-sort-by</sortKey>
    </myqueue>
</queue>
```

7.2.5.1.3. LVQ

To define a LVQ, add a `lvq` element with the value `true`. Without any further configuration this will define an LVQ which uses the JMS message property `qpid.LVQ_key` as the key for replacement.

Example 7.4. Configuring a LVQ queue

```
<queue>
    <name>myqueue</name>
    <myqueue>
        <exchange>amq.direct</exchange>
        <lvq>true</lvq>
    </myqueue>
</queue>
```

If you wish to define your own property then you can do so using the `lvqKey` element.

Example 7.5. Configuring a LVQ queue with custom message property name

```
<queue>
    <name>myqueue</name>
    <myqueue>
        <exchange>amq.direct</exchange>
        <lvq>true</lvq>
        <lvqKey>ISIN</lvqKey>
    </myqueue>
</queue>
```

7.2.5.2. Using JMX or AMQP

To create a priority, sorted or LVQ queue programmatically from JMX or using a Qpid extension to JMS, pass the appropriate queue-declare arguments.

Table 7.1. Queue-declare arguments understood for priority, sorted and LVQ queues

Queue type	Argument name	Argument name	Argument Description
priority	priorities	java.lang.Integer	Specifies a priority queue with given number priorities
sorted	qpid.queue_sort_key	java.lang.String	Specifies sorted queue with given message property used to sort the entries

Queue type	Argument name	Argument name	Argument Description
lvq	qpid.last_value_queue_key	java.lang.String	Specifies lvq queue with given message property used to conflate the entries

The following example illustrates the creation of the a LVQ queue from a javax.jms.Session object. Note that this utilises a Qpid specific extension to JMS and involves casting the session object back to its Qpid base-class.

Example 7.6. Creation of an LVQ using the Qpid extension to JMS

```
Map<String,Object> arguments = new HashMap<String, Object>();
arguments.put("qpid.last_value_queue_key","ISIN");
((AMQSession<?,?>) session).createQueue(queueName, autoDelete, durable, exclusive,
```

The following example illustrates the creation of the sorted queue from a the JMX interface using the ManagedBroker interface.

Example 7.7. Creation of a sorted queue using JMX

```
Map<String, Object> environment = new HashMap<String, Object>();
environment.put(JMXConnector.CREDENTIALS, new String[] {"admin","password"});
// Connect to service
JMXServiceURL url =  new JMXServiceURL("service:jmx:rmi:///jndi/rmi://localhost:89
JMXConnector jmxConnector = JMXConnectorFactory.connect(url, environment);
MBeanServerConnection mbsc =  jmxConnector.getMBeanServerConnection();
// Object name for ManagedBroker for virtualhost myvhost
ObjectName objectName = new ObjectName("org.apache.qpid:type=VirtualHost.VirtualHo
// Get the ManagedBroker object
ManagedBroker managedBroker = JMX.newMBeanProxy(mbsc, objectName, ManagedBroker.cl

// Create the queue passing arguments
Map<String,Object> arguments = new HashMap<String, Object>();
arguments.put("qpid.queue_sort_key","myheader");
managedBroker.createNewQueue("myqueue", null, true, arguments);
```

7.2.6. Messaging Grouping

The broker allows messaging applications to classify a set of related messages as belonging to a group. This allows a message producer to indicate to the consumer that a group of messages should be considered a single logical operation with respect to the application.

The broker can use this group identification to enforce policies controlling how messages from a given group can be distributed to consumers. For instance, the broker can be configured to guarantee all the messages from a particular group are processed in order across multiple consumers.

For example, assume we have a shopping application that manages items in a virtual shopping cart. A user may add an item to their shopping cart, then change their mind and remove it. If the application sends an *add* message to the broker, immediately followed by a *remove* message, they will be queued in the proper order - *add*, followed by *remove*.

However, if there are multiple consumers, it is possible that once a consumer acquires the *add* message, a different consumer may acquire the *remove* message. This allows both messages to be processed in

parallel, which could result in a "race" where the *remove* operation is incorrectly performed before the *add* operation.

7.2.6.1. Grouping Messages

In order to group messages, the application would designate a particular message header as containing a message's *group identifier*. The group identifier stored in that header field would be a string value set by the message producer. Messages from the same group would have the same group identifier value. The key that identifies the header must also be known to the message consumers. This allows the consumers to determine a message's assigned group.

The header that is used to hold the group identifier, as well as the values used as group identifiers, are totally under control of the application.

7.2.6.2. The Role of the Broker in Message Grouping

The broker will apply the following processing on each grouped message:

- Enqueue a received message on the destination queue.
- Determine the message's group by examining the message's group identifier header.
- Enforce *consumption ordering* among messages belonging to the same group. *Consumption ordering* means one of two things depending on how the queue has been configured.
  - In default mode, each group is assigned to a consumer for the lifetime of the consumer.
  - In C++ compatibility mode (which gives the same behaviour as the C++ Qpid Broker), the Broker enforces a looser guarantee, nameley that all the *currently unacknowledged messages* in a group will be sent to the same consumer. This means that only one consumer can be processing messages from a particular group at a given time. When the consumer acknowledges all of its acquired messages, then the broker *may* pass the next pending message from that group to a different consumer.

The absence of a value in the designated header field for grouping as treated as indicative of a lack of desire for the message to be grouped. Messages with such a lack of a value will be distributed to any available consumer.

Note that message grouping has no effect on queue browsers.

Note well that distinct message groups would not block each other from delivery. For example, assume a queue contains messages from two different message groups - say group "A" and group "B" - and they are enqueued such that "A"'s messages are in front of "B". If the first message of group "A" is in the process of being consumed by a client, then the remaining "A" messages are blocked, but the messages of the "B" group are available for consumption by other consumers - even though it is "behind" group "A" in the queue.

7.2.6.3. Broker Message Grouping Configuration

In order for the broker to determine a message's group, the key for the header that contains the group identifier must be provided to the broker via configuration. This is done on a per-queue basis, when the queue is first configured.

This means that message group classification is determined by the message's destination queue.

Specifically, the queue "holds" the header key that is used to find the message's group identifier. All messages arriving at the queue are expected to use the same header key for holding the identifer. Once

the message is enqueued, the broker looks up the group identifier in the message's header, and classifies the message by its group.

Message group support is specified by providing one or more of the following settings in the arguments map that is used when declaring the queue (e.g. when calling `AMQSession.createQueue()`).

Table 7.2. Queue Declare Message Group Configuration Arguments

Key	Value
qpid.group_header_key	The name of the message header that holds the group identifier value. The values in this header may be of any supported type (i.e. not just strings).
qpid.shared_msg_group	Provide a value of "1" to switch on C++ compatibility mode

It is important to note that there is no need to provide the actual group identifer values that will be used. The broker learns these values as messages are recieved. Also, there is no practical limit - aside from resource limitations - to the number of different groups that the broker can track at run time.

7.2.7. Using low pre-fetch with special queue types

Qpid clients receive buffered messages in batches, sized according to the pre-fetch value. The current default is 500.

However, if you use the default value you will probably *not* see desirable behaviour when using priority, sorted, lvq or grouped queues. Once the broker has sent a message to the client its delivery order is then fixed, regardless of the special behaviour of the queue.

For example, if using a priority queue and a prefetch of 100, and 100 messages arrive with priority 2, the broker will send these messages to the client. If then a new message arrives will priority 1, the broker cannot leap frog messages of lower priority. The priority 1 will be delivered at the front of the next batch of messages to be sent to the client.

So, you need to set the prefetch values for your client (consumer) to make this sensible. To do this set the Java system property `max_prefetch` on the client environment (using -D) before creating your consumer.

A default for all client connections can be set via a system property:

```
-Dmax_prefetch=1
```

The prefetch can be also be adjusted on a per connection basis by adding a `maxprefetch` value to the Connection URLs [../../Programming-In-Apache-Qpid/html/QpidJNDI.html#section-jms-connection-url]

```
amqp://guest:guest@client1/development?maxprefetch='1'&brokerlist='tcp://localhost
```

Setting the Qpid pre-fetch to 1 will give exact queue-type semantics as perceived by the client however, this brings a performance cost. You could test with a slightly higher pre-fetch to trade-off between throughput and exact semantics.

Chapter 8. Stores

8.1. Memory Store

The Java broker has an in-memory message store implementation. This section will detail configuration for using the MemoryMessageStore.

Note: when using this store, the broker will store both persistent and non-persistent messages in memory, which is to say that neither will be available following a broker restart, and the ability to store new messages will be entirely constrained by the JVM heap size.

8.1.1. Configuration

In order to use the MemoryMessageStore, you must configure it for each VirtualHost desired by updating the store element to specify the associated store class, as shown below.

Example 8.1. Configuring a VirtualHost to use the MemoryMessageStore

```
<virtualhosts>
  <virtualhost>
    <name>vhostname</name>
    <vhostname>
      <store>
        <class>org.apache.qpid.server.store.MemoryMessageStore</class
      </store>
      ...
    </vhostname>
  </virtualhost>
</virtualhosts>
```

8.2. Derby Store

The Java broker has a message store implementation backed by Apache Derby. This section will detail configuration for using the DerbyMessageStore.

8.2.1. Configuration

In order to use the DerbyMessageStore, you must configure it for each VirtualHost desired by updating the store element to specify the associated store class and provide a directory location for the data to be written, as shown below.

Example 8.2. Configuring a VirtualHost to use the DerbyMessageStore

```
<virtualhosts>
  <virtualhost>
    <name>vhostname</name>
    <vhostname>
      <store>
        <class>org.apache.qpid.server.store.DerbyMessageStore</class>
        <environment-path>${QPID_WORK}/derbystore/vhostname</environment-path>
      </store>
      ...
    </vhostname>
  </virtualhost>
</virtualhosts>
```

8.3. SQL Store

8.4. BDB Store

The Java broker has an *optional* message store implementation backed by Oracle BDB JE. This section will detail where to download the optional dependency from, how to add it to the broker installation, and provide an example configuration for using the BDBMessageStore.

8.4.1. Oracle BDB JE download

The BDB based message store is optional due to its dependency on Oracle BDB JE, which is distributed under the Sleepycat licence. As a result of this, the dependency cant be distributed by the Apache Qpid project as part of the broker release package.

If you wish to use the BDBMessageStore, then you must download the Oracle BDB JE 5.0.73 release from the Oracle website. [http://www.oracle.com/technetwork/products/berkeleydb/downloads/index.html?ssSourceSiteId=ocomen]

The download has a name in the form je-5.0.73.tar.gz. It is recommended that you confirm the integrity of the download by verifying the MD5.

8.4.2. Oracle BDB JE jar installation

If you wish to use the BDBMessageStore, copy the je-5.0.73.jar from within the release downloaded above into the 'opt' sub-directory of the brokers 'lib' directory.

```
Unix:
cp je-5.0.73.jar qpid-broker-0.22/lib/opt
```

```
Windows:
copy je-5.0.73.jar qpid-broker-0.22\lib\opt
```

8.4.3. Configuration

In order to use the BDBMessageStore, you must configure it for each VirtualHost desired by updating the store element to specify the associated store class and provide a directory location for the data to be written, as shown below.

Example 8.3. Configuring a VirtualHost to use the BDBMessageStore

```
<virtualhosts>
  <virtualhost>
    <name>vhostname</name>
    <vhostname>
      <store>
        <class>org.apache.qpid.server.store.berkeleydb.BDBMessageStore</class>
        <environment-path>${QPID_WORK}/bdbstore/vhostname</environment-path>
      </store>
      ...
    </vhostname>
  </virtualhost>
</virtualhosts>
```

8.5. High Availability BDB Store

The Java broker has an *optional* High Availability message store implementation backed by Oracle BDB JE HA. This section references information on where to download the optional dependency from, how to add it to the broker installation, and how to configure the BDBHAMessageStore.

For more detailed information about use of this store, see Chapter 12, *High Availability*.

8.5.1. Oracle BDB JE download

For details, see Section 8.4.1, “Oracle BDB JE download”.

8.5.2. Oracle BDB JE jar installation

For details, see Section 8.4.2, “Oracle BDB JE jar installation”.

8.5.3. Configuration

In order to use the BDBHAMessageStore, you must configure it for each VirtualHost desired by updating the store element to specify the associated store class, provide a directory location for the data to be written, and configure the replication group and policies used by BDB JA HA.

A general configuration example is shown here, however it is strongly recommended you examine the wider context of Chapter 12, *High Availability* for a fuller discussion of the various configuration options and how to use them.

Chapter 9. Configuring And Managing

9.1. Config Files

This section shows how to configure and manage broker.

9.1.1. Configuration file

Broker can be configured using XML configuration files. By default, broker is looking for configuration file at ${QPID_HOME}/etc/config.xml. The default configuration location can be overridden by specifying command line option *-c <path to configuration>* on broker start up.

9.1.2. Management Configuration

Management interfaces can be configured in *management* section of broker configuration file. The example of the management section is provided below.

Example 9.1. Management configuration

```
<broker>
...
    <management>
        <enabled>true</enabled>
        <jmxport>
            <registryServer>8999</registryServer>
        </jmxport>
        <ssl>
            <enabled>false</enabled>
            <keyStorePath>${conf}/qpid.keystore</keyStorePath>
            <keyStorePassword>password</keyStorePassword>
        </ssl>
        <http>
            <enabled>true</enabled>
        </http>
        <https>
            <enabled>false</enabled>
        </https>
    </management>
...
</broker>
```

9.1.3. JMX Management Configuration

JMX management can be configured in *management* section of broker configuration file.

An *enabled* element in the *management* section is used to enable or disable the JMX interfaces. Setting it to *true* causes the broker to start the management plugin if such is available on the broker classpath.

JMX management requires two ports which can be configured in *jmxport* sub-section of *management*:

- RMI port (8999 by default) can be configured in an element *jmxport/registryServer*
- Connector port can be configured in an element *jmxport/connectorServer*. If configuration element *connectorServer* is not provided than the connector port defaults to *100 + registryServer port*.

Example 9.2. Enabling JMX Management and configuring JMX ports

```
<broker>
...
<management>
    <enabled>true</enabled>                          1
    <jmxport>
        <registryServer>7999</registryServer>        2
        <connectorServer>7998</connectorServer>      3
    </jmxport>
</management>
...
</broker>
```

In the snippet above the following is configured:

1 Enable JMX management
2 Set RMI port to 7999
3 Set connector port to 7998

SSL can be configured to use on the connector port in the sub-section *ssl* of the *management* section. See Section 9.1.4, “Management SSL key store configuration” for details.

In order to use SSL with JMX management an element *ssl/enabled* needs to be set to *true*.

9.1.4. Management SSL key store configuration

This section describes how to configure the key store to use in SSL connections in both JMX and Web management interfaces.

The following examples demonstrates how to configure keystore for management

Example 9.3. Management key store configuration

```
<broker>
...
<management>
...
    <ssl>
        <enabled>true</enabled>                                  1
        <keyStorePath>${conf}/qpid.keystore</keyStorePath>       2
        <keyStorePassword>password</keyStorePassword>            3
    </ssl>
...
</management>
...
</broker>
```

1 Enable SSL on JMX connector port only. This setting does not effect the web management interfaces.
2 Set path to the key store file
3 Set keystore password

9.1.5. Web Management Configuration

Web management can be configured in *management* section of broker configuration file.

Sub-section *http* is used to enable web management on http port.

Sub-section *https* is used to enable web management on https port.

The following example shows how to configure http and https ports

Example 9.4. Enabling web management

```
<broker>
...
<management>
...
    <http>
        <enabled>true</enabled>                          1
        <port>9090</port>                                2
        <basic-auth>false</basic-auth>                   3
        <sasl-auth>true</sasl-auth>                      4
        <session-timeout>600</session-timeout>           5
    </http>

    <https>
        <enabled>true</enabled>                          6
        <port>9443</port>                                7
        <sasl-auth>true</sasl-auth>                      8
        <basic-auth>true</basic-auth>                    9
    </https>
...
</management>
...
</broker>
```

1 Enable web management on http port. Default is true.
2 Set web management http port to 9090. Default is 8080.
3 Disable basic authentication on http port for REST services only. Default is false.
4 Enable SASL authentication on http port for REST services and web console. Default is true.
5 Set session timeout in seconds. Default is 15 minutes.
6 Enable web management on https port. Default is false.
7 Set web management https port to 9443. Default is 8443.
8 Enable SASL authentication on https port for REST services and web console. Default is true.
9 Enable basic authentication on https port for REST services only. Default is true.

Note

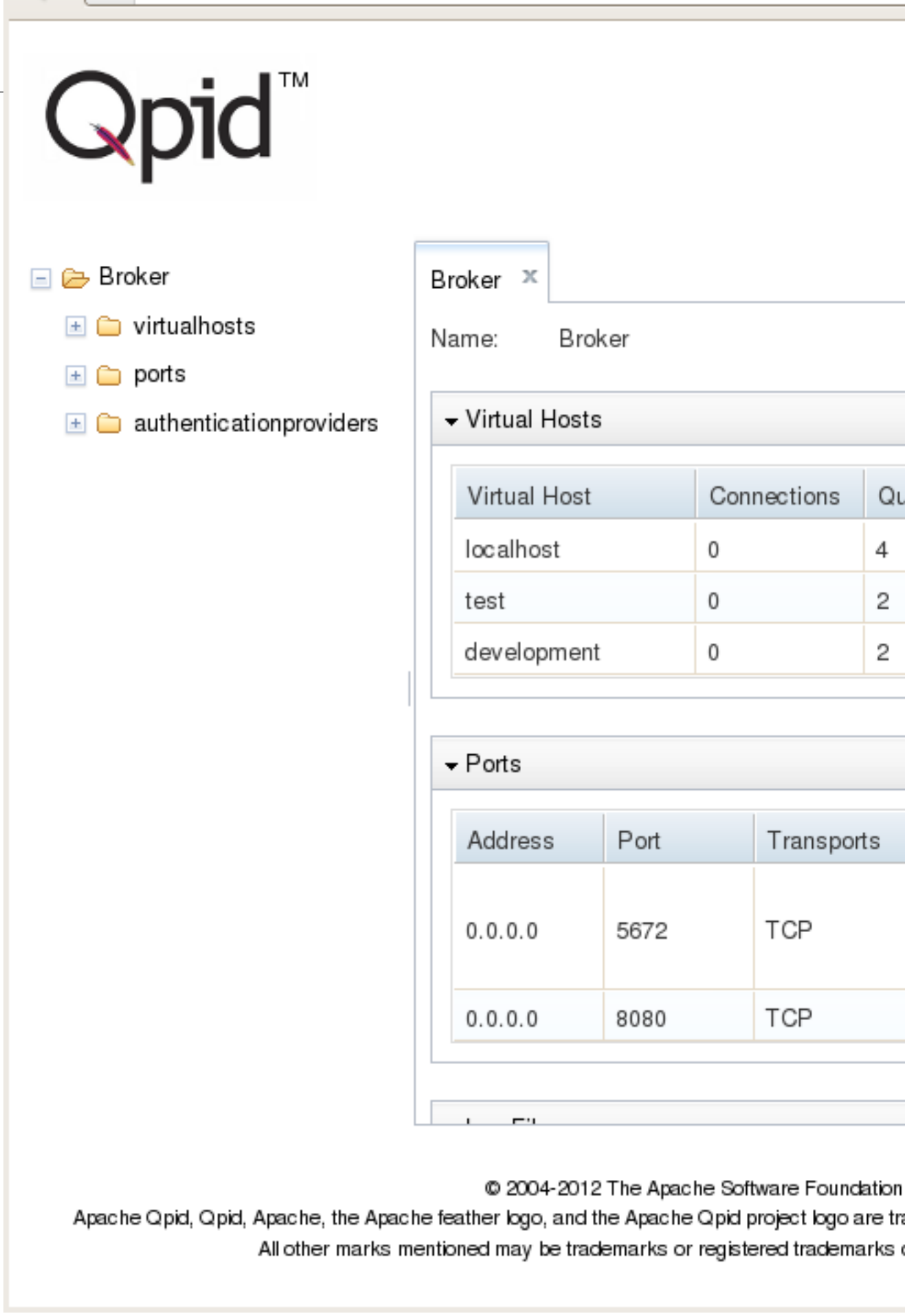
Please configure the keystore to use with the https web management port. See Section 9.1.4, “Management SSL key store configuration” for details.

9.2. Web Console

If web management is enabled (see Section 9.1.5, “Web Management Configuration”) the web management console can be accessed from web browser using URL http(s)://<hostname>:<port>/management, where

- *hostname* is the broker host
- *port* is the broker port(either http or https)

The page like following is displayed on navigation to the management URL.


Qpid™
Broker
virtualhosts
ports
authenticationproviders
Broker
Name: Broker
Virtual Hosts
Virtual Host
Connections
localhost
0
4
test
0
2
development
0
2
Ports
Address
Port
Transports
0.0.0.0
5672
TCP
0.0.0.0
8080
TCP
© 2004-2012 The Apache Software Foundation
Apache Qpid, Qpid, Apache, the Apache feather logo, and the Apache Qpid project logo are tr
All other marks mentioned may be trademarks or registered trademarks

9.3. REST API

9.3.1. REST API Overview

This section provides an overview of REST management API.

If web management is enabled (see Section 9.1.5, “Web Management Configuration”) the REST API can be used to monitor and manage the broker instance.

The Qpid broker REST services support traditional REST model which uses the GET method requests to retrieve the information about broker configured objects, DELETE method requests to delete the configured object, PUT to create the configured object and POST to update the configured objects.

The table below lists the available REST services with brief description how they can be used.

Table 9.1. Rest services

Rest service URL	Description	GET	PUT	POST	DELETE
/rest/broker	Rest service to manage broker instance	Retrieves the details of broker configuration	Not implemented yet	Not implemented yet	Not implemented yet
/rest/authenticationprovider /rest/authenticationprovider/<authentication provider name>	Rest service to manage authentication providers on the broker	Retrieves the details about authentication providers	Not implemented yet	Not implemented yet	Not implemented yet
/rest/user /rest/user/<authentication provider name>/<user name>	Rest service to manage user account	Retrieves the details about user account	Creates user account	Updates user password	Deletes user account
/rest/groupprovider /rest/groupprovider/<group provider name>	Rest service to manage group providers	Retrieves the details about group provider(s)	Not implemented yet	Not implemented yet	Not implemented yet
/rest/group /rest/group/<group provider name>/<group name>	Rest service to manage user group	Retrieves the details about user group	Creates group	Not implemented yet	Deletes group
/rest/groupmember	Rest service to manage group member(s)	Retrieves the details about	Add user to group	Not implemented yet	Deletes user from group

Rest service URL	Description	GET	PUT	POST	DELETE
/rest/ groupmember/ <group provider name >/<group name>/<user name>		group member(s)			
/rest/port /rest/port/<port name>	Rest service to manage broker ports(s)	Retrieves the details about the broker port(s)	Not implemented yet	Not implemented yet	Not implemented yet
/rest/port /rest/port/<port name>	Rest service to manage broker ports(s)	Retrieves the details about the broker port(s)	Not implemented yet	Not implemented yet	Not implemented yet
/rest/queue /rest/queue/ <virtual host name>/>queue name>	Rest service to manage queue(s)	Retrieves the details about the queue(s)	Creates queue	Not implemented yet	Deletes queue
/rest/exchange /rest/exchange/ <virtual host name>/ <exchange name>	Rest service to manage exchange(s)	Retrieves the details about the exchange(s)	Creates exchange	Not implemented yet	Deletes exchange
/rest/binding /rest/binding/ <virtual host name>/ <exchange name>/<queue name>/ <binding name>	Rest service to manage binding(s)	Retrieves the details about the binding(s)	Binds a queue to an exchange	Not implemented yet	Deletes binding
/rest/connection /rest/ connection/ <virtual host name>/ <connection name>	Rest service to manage connection(s)	Retrieves the details about the connection(s)	Not implemented yet	Not implemented yet	Not implemented yet
/rest/session /rest/session/ <virtual host name>/ <connection	Rest service to manage session(s)	Retrieves the details about the session(s)	Not implemented yet	Not implemented yet	Not implemented yet

Rest service URL	Description	GET	PUT	POST	DELETE
name>/<session name>					
/rest/message/*	Rest service to manage messages(s)	Retrieves the details about the messages(s)	Not implemented yet	Copies, moves messages	Deletes messages
/rest/message-content/*	Rest service to retrieve message content	Retrieves the message content	Not implemented yet	Not implemented yet	Not implemented yet
/rest/logrecords	Rest service to retrieve broker logs	Retrieves the broker logs	Not implemented yet	Not implemented yet	Not implemented yet
/rest/sasl	Sasl authentication	Retrieves user current authentication status and broker supported SASL mechanisms	Authenticates user using supported SASL mechanisms	Not implemented yet	Not implemented yet
/rest/logout	Log outs	Log outs user	Not implemented yet	Not implemented yet	Not implemented yet

Rest URL are hierarchical. It is permitted to replace rest URL elements with an "asterisks" in GET requests to denote all object of a particular type. Additionally, trailing object type in the URL hierarchy can be omitted. In this case GET request will return all of the object underneath of the current object.

For example, for binding URL http://localhost:8080/rest/binding/<vhost>/<exchange>/<queue>/<binding> replacing of <*exchange*> with "asterisks" (http://localhost:8080/rest/binding/<vhost>/*/<queue>/<binding>) will result in the GET response containing the list of bindings for all of the exchanges in the virtual host having the given name and given queue. If <*binding*> and <*queue*> are omitted in binding REST URL (http://localhost:8080/rest/binding/<vhostname>/<exchangename>) the GET request will result in returning all bindings for all queues for the given exchange in the virtual host.

Example 9.5. Examples of queue creation using curl:

```
#create a durable queue
curl -X PUT  -d '{"durable":true}' http://localhost:8080/rest/queue/<vhostname>/<q
#create a durable priority queue
curl -X PUT  -d '{"durable":true,"type":"priority"}' http://localhost:8080/rest/qu
```

Example 9.6. Example of binding a queue to an exchange using curl

```
curl  -X PUT  -d '{}' http://localhost:8080/rest/binding/<vhostname>/<exchangename
```

Qpid broker web management console calls rest interfaces internally to manage the broker.

9.4. JMX

9.5. Other Tooling

Chapter 10. Security

10.1. Users And Groups

10.2. Configuring Group Providers

The Java broker utilises GroupProviders to allow assigning users to groups for use in ACLs. Following authentication by a given Authentication Provider, the configured Group Providers are consulted to allowing assignment of GroupPrincipals for a given authenticated user.

10.2.1. FileGroupManager

The FileGroupManager allows specifying group membership in a flat file on disk, and is also exposed for inspection and update through the brokers HTTP management interface.

To enable the FileGroupManager, add the following configuration to the config.xml, adjusting the groupFile attribute value to match your desired groups file location.

```
...
<security>
    <file-group-manager>
        <attributes>
          <attribute>
            <name>groupFile</name>
             <value>${conf}/groups</value>
          </attribute>
        </attributes>
    </file-group-manager>
</security>
...
```

10.2.1.1. File Format

The groups file has the following format:

```
# <GroupName>.users = <comma deliminated user list>
# For example:

administrators.users = admin,manager
```

Only users can be added to a group currently, not other groups. Usernames can't contain commas.

Lines starting with a '#' are treated as comments when opening the file, but these are not preserved when the broker updates the file due to changes made through the management interface.

10.3. Authentication Providers

In order to successfully establish a connection to the Java Broker, the connection must be authenticated. The Java Broker supports a number of different authentication schemes, each with its own "authentication manager". Each of these are outlined below, along with details of using more than one at a time.

10.3.1. Password File

TODO

10.3.2. LDAP

LDAP authentication can be configured using the <simple-ldap-auth-manager> element within the <security> section. An example of how to configure this is shown below. Please note this example also configures an unused <pd-auth-manager> to use an empty password file, this is a workaround for an issue relating to registration of security providers.

NOTE: When using LDAP authentication, you must also use SSL on the brokers AMQP messaging and JMX/HTTP management ports in order to protect passwords during transmission to the broker.

Example 10.1. Configuring LDAP authentication

```
<security>
  <default-auth-manager>SimpleLDAPAuthenticationManager</default-auth-manager>
  <simple-ldap-auth-manager>
    <provider-url>ldaps://example.com:636/</provider-url>
    <search-context>dc=example\,dc=com</search-context>
    <search-filter>(uid={0})</search-filter>
  </simple-ldap-auth-manager>

  <!-- Unused pd-auth-manager, a workaround to register the necessary security pro
  <pd-auth-manager>
    <principal-database>
      <class>org.apache.qpid.server.security.auth.database.PlainPasswordFilePrinci
      <attributes>
        <attribute>
          <name>passwordFile</name>
          <value>${conf}/emptyPasswdFile</value>
        </attribute>
      </attributes>
    </principal-database>
  <pd-auth-manager>
  ...
</security>
```

The authentication manager first connects to the ldap server anonymously and searches for the ldap entity which is identified by the username provided over SASL. Essentially the authentication manager calls DirContext.search(Name name, String filterExpr, Object[] filterArgs, SearchControls cons) with the values of search-context and search-filter as the first two arguments, and the username as the only element in the array which is the third argument.

If the search returns a name from the LDAP server, the AuthenticationManager then attempts to login to the ldap server with the given name and the password.

If the URL to open for authentication is different to that for the search, then the authentication url can be overridden using <provider-auth-url> in addition to providing a <provider-url>. Note that the URL used for authentication should use ldaps:// since passwords will be being sent over it.

By default com.sun.jndi.ldap.LdapCtxFactory is used to create the context, however this can be overridden by specifying <ldap-context-factory> in the configuration.

10.3.3. Kerberos

Kereberos Authentication is configured using the <kerberos-auth-manager> element within the <security> section. When referencing from the default-auth-manager or port-mapping sections, its name is KerberosAuthenticationManager.

Since Kerberos support only works where SASL authentication is available (e.g. not for JMX authentication) you may wish to also include an alternative Authentication Manager configuration, and use this for other ports:

Example 10.2. Configuring Kerberos authentication

```
<security>
  <pd-auth-manager>
    <principal-database>
      <class>org.apache.qpid.server.security.auth.database.PlainPasswordFilePrinci
      <attributes>
        <attribute>
          <name>passwordFile</name>
          <value>${conf}/passwd</value>
        </attribute>
      </attributes>
    </principal-database>
  </pd-auth-manager>
  <kerberos-auth-manager/>
  <default-auth-manager>PrincipalDatabaseAuthenticationManager</default-auth-manag
  <port-mappings>
    <port-mapping>
      <port>5672</port>
      <auth-manager>KerberosAuthenticationManager</auth-manager>
    </port-mapping>
  </port-mappings>
  ...
</security>
```

Configuration of kerberos is done through system properties (there doesn't seem to be a way around this unfortunately).

```
    export QPID_OPTS=-Djavax.security.auth.useSubjectCredsOnly=false -Djava.securi
    ${QPID_HOME}/bin/qpid-server
```

Where qpid.conf would look something like this:

```
com.sun.security.jgss.accept {
    com.sun.security.auth.module.Krb5LoginModule required
    useKeyTab=true
    storeKey=true
    doNotPrompt=true
    realm="EXAMPLE.COM"
    useSubjectCredsOnly=false
    kdc="kerberos.example.com"
```

```
    keyTab="/path/to/keytab-file"
    principal="<name>/<host>";
};
```

Where realm, kdc, keyTab and principal should obviously be set correctly for the environment where you are running (see the existing documentation for the C++ broker about creating a keytab file).

Note: You may need to install the "Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files" appropriate for your JDK in order to get Kerberos support working.

10.3.4. External (SSL Client Certificates)

When requiring SSL Client Certificates be presented the ExternalAuthenticationManager can be used, such that the user is authenticated based on trust of their certificate alone, and the X500Principal from the SSL session is then used as the username for the connection, instead of also requiring the user to present a valid username and password.

The ExternalAuthenticationManager may be enabled by adding an empty <external-auth-manager> element to the <security> section, as shown below. When referencing it from the default-auth-manager or port-mapping sections, its name is ExternalAuthenticationManager.

Note: The ExternalAuthenticationManager should typically only be used on the AMQP ports, in conjunction with SSL client certificate authentication. It is not intended for other uses such as the JMX management port and will treat any non-sasl authentication processes on these ports as successfull with the given username. As such you should include another Authentication Manager for use on non-AMQP ports, as is done in the example below. Perhaps the only exception to this would be where the broker is embedded in a container that is itself externally protecting the HTTP interface and then providing the remote users name.

Example 10.3. Configuring external authentication (SSL client auth)

```
<security>
  <pd-auth-manager>
    <principal-database>
      <class>org.apache.qpid.server.security.auth.database.PlainPasswordFilePrinci
      <attributes>
        <attribute>
          <name>passwordFile</name>
          <value>${conf}/passwd</value>
        </attribute>
      </attributes>
    </principal-database>
  </pd-auth-manager>
  <external-auth-manager/>
  <default-auth-manager>PrincipalDatabaseAuthenticationManager</default-auth-manag
  <port-mappings>
    <port-mapping>
      <port>5672</port>
      <auth-manager>ExternalAuthenticationManager</auth-manager>
    </port-mapping>
  </port-mappings>
  ...
</security>
```

10.3.5. Anonymous

The AnonymousAuthenticationManager will allow users to connect with or without credentials and result in their identification on the broker as the user ANONYMOUS. It may be enabled by adding an empty anonymous-auth-manager element to the security configuration section, as shown below.

Example 10.4. Configuring anonymous authentication

```
<security>
  <anonymous-auth-manager/>
  ...
</security>
```

When referencing it from the default-auth-manager or port-mapping sections, its name is AnonymousAuthenticationManager.

10.3.6. Configuring multiple Authentication Providers

Different managers may be used on different ports. Each manager has its own configuration element, the presence of which within the <security> section denotes the use of that authentication provider. Where only one such manager is configured, it will be used on all ports (including JMX and HTTP). Where more than one authentication manager is configured the configuration must define which is the "default", and (if required) the mapping of non-default authentication managers to other ports.

The following configuration sets up three authentication managers, using a password file as the default (e.g. for the JMX and HTTP ports), Kerberos on port 5672 (the regular AMQP port) and Anonymous on port 5673 (e.g a second AMQP port the broker could have been configured with).

Example 10.5. Configuring multiple (per-port) authentication schemes

```
<security>
  <pd-auth-manager>
    <principal-database>
      <class>org.apache.qpid.server.security.auth.database.PlainPasswordFilePrinci
      <attributes>
        <attribute>
          <name>passwordFile</name>
          <value>${conf}/passwd</value>
        </attribute>
      </attributes>
    </principal-database>
  </pd-auth-manager>
  <kerberos-auth-manager>
    <auth-name>sib</auth-name>
  </kerberos-auth-manager>
  <anonymous-auth-manager/>
  <default-auth-manager>PrincipalDatabaseAuthenticationManager</default-auth-manag
  <port-mappings>
    <port-mapping>
      <port>5672</port>
        <auth-manager>KerberosAuthenticationManager</auth-manager>
      </port-mapping>
    <port-mapping>
      <port>5673</port>
        <auth-manager>AnonymousAuthenticationManager</auth-manager>
    </port-mapping>
  </port-mappings>
  ...
</security>
```

10.4. Access Control Lists

In Qpid, Access Control Lists (ACLs) specify which actions can be performed by each authenticated user. To enable, the <acl/> element is used within the <security/> element of the configuration XML. In the Java Broker, the ACL may be imposed broker wide or applied to individual virtual hosts. The <acl/> configuration references a text file containing the ACL rules. By convention, this file should have a .acl extension.

10.4.1. Enabling ACLs

To apply an ACL broker-wide, add the following to the config.xml (assuming that *conf* has been set to a suitable location such as ${QPID_HOME}/etc):

```
<broker>
  ...
  <security>
    ...
    <acl>${conf}/broker.acl</acl>
  </security>
```

```
</broker>
```

To apply an ACL on a single virtualhost named *test*, add the following to the config.xml:

```
<virtualhost>
  ...
  <name>test</name>
  <test>
    ...
    <security>
      <acl>${conf}/vhost_test.acl</acl>
    </security>
  </test>
</virtualhost>
```

10.4.2. Writing .acl files

The ACL file consists of a series of rules associating behaviour for a user or group. Use of groups can serve to make the ACL file more concise. See Configuring Group Providers for more information on defining groups.

Each ACL rule grants or denies a particular action on an object to a user/group. The rule may be augmented with one or more properties, restricting the rule's applicability.

```
ACL ALLOW alice CREATE QUEUE              # Grants alice permission to creat
ACL DENY bob CREATE QUEUE name="myqueue"  # Denies bob permission to create
```

The ACL is considered in strict line order with the first matching rule taking precedence over all those that follow. In the following example, if the user bob tries to create an exchange "myexch", the operation will be allowed by the first rule. The second rule will never be considered.

```
ACL ALLOW bob ALL EXCHANGE
ACL DENY bob CREATE EXCHANGE name="myexch"  # Dead rule
```

If the desire is to allow bob to create all exchanges except "myexch", order of the rules must be reversed:

```
ACL DENY bob CREATE EXCHANGE name="myexch"
ACL ALLOW bob ALL EXCHANGE
```

All ACL files end with an implict rule denying all operations to all users. It is as if each file ends with

```
ACL DENY ALL ALL
```

If instead you wish to *allow* all operations other than those controlled by earlier rules, add

```
ACL ALLOW ALL ALL
```

to the bottom of the ACL file.

When writing a new ACL, a good approach is to begin with an .acl file containing only

```
ACL DENY-LOG ALL ALL
```

which will cause the Broker to deny all operations with details of the denial logged to the Qpid log file. Build up the ACL rule by rule, gradually working through the use-cases of your system. Once the ACL is complete, consider switching the DENY-LOG actions to DENY to improve performamce and reduce log noise.

ACL rules are very powerful: it is possible to write very granular rules specifying many broker objects and their properties. Most projects probably won't need this degree of flexibility. A reasonable approach is to choose to apply permissions at a certain level of abstraction (e.g. QUEUE) and apply them consistently across the whole system.

10.4.3. Syntax

ACL rules follow this syntax:

```
ACL {permission} {<group-name>|<user-name>>|ALL} {action|ALL} [object|ALL] [p
```

Comments may be introduced with the hash (#) character and are ignored. Long lines can be broken with the slash (\) character.

```
# A comment
ACL ALLOW admin CREATE ALL # Also a comment
ACL DENY guest \
ALL ALL   # A broken line
```

Table 10.1. List of ACL permission

ALLOW	Allow the action
ALLOW-LOG	Allow the action and log the action in the log
DENY	Deny the action
DENY-LOG	Deny the action and log the action in the log

Table 10.2. List of ACL actions

CONSUME	Applied when subscriptions are created
PUBLISH	Applied on a per message basis on publish message transfers
CREATE	Applied when an object is created, such as bindings, queues, exchanges
ACCESS	Applied when an object is read or accessed
BIND	Applied when queues are bound to exchanges
UNBIND	Applied when queues are unbound from exchanges

DELETE	Applied when objects are deleted
PURGE	Applied when purge the contents of a queue
UPDATE	Applied when an object is updated

Table 10.3. List of ACL objects

VIRTUALHOST	A virtualhost (Java Broker only)
MANAGEMENT	Management - for web and JMX (Java Broker only)
QUEUE	A queue
EXCHANGE	An exchange
USER	A user (Java Broker only)
GROUP	A group (Java Broker only)
METHOD	Management or agent or broker method (Java Broker only)
LINK	A federation or inter-broker link (not currently used in Java Broker)
BROKER	The broker (not currently used in Java Broker)

Table 10.4. List of ACL properties

name	String. Object name, such as a queue name, exchange name or JMX method name.
durable	Boolean. Indicates the object is durable
routingkey	String. Specifies routing key
passive	Boolean. Indicates the presence of a *`passive`* flag
autodelete	Boolean. Indicates whether or not the object gets deleted when the connection is closed
exclusive	Boolean. Indicates the presence of an *`exclusive`* flag
temporary	Boolean. Indicates the presence of an *`temporary`* flag
type	String. Type of object, such as topic, fanout, or xml
alternate	String. Name of the alternate exchange
queuename	String. Name of the queue (used only when the object is something other than *`queue`*
component	String. JMX component name (Java Broker only)
schemapackage	String. QMF schema package name (Not used in Java Broker)
schemaclass	String. QMF schema class name (Not used in Java Broker)
from_network	Comma-separated strings representing IPv4 address ranges. Intended for use in ACCESS VIRTUALHOST rules to apply firewall-like restrictions.

	The rule matches if any of the address ranges match the IPv4 address of the messaging client. The address ranges are specified using either Classless Inter-Domain Routing notation (e.g. 192.168.1.0/24; see RFC 4632 [http://tools.ietf.org/html/rfc4632]) or wildcards (e.g. 192.169.1.*). Java Broker only.
from_hostname	Comma-separated strings representing hostnames, specified using Perl-style regular expressions, e.g. .*\.example\.company\.com Intended for use in ACCESS VIRTUALHOST rules to apply firewall-like restrictions. The rule matches if any of the patterns match the hostname of the messaging client. To look up the client's hostname, Qpid uses Java's DNS support, which internally caches its results. You can modify the time-to-live of cached results using the *.ttl properties described on the Java Networking Properties [http://docs.oracle.com/javase/6/docs/technotes/guides/net/properties.html] page. For example, you can either set system property sun.net.inetaddr.ttl from the command line (e.g. export QPID_OPTS="-Dsun.net.inetaddr.ttl=0") or networkaddress.cache.ttl in $JAVA_HOME/lib/security/java.security. The latter is preferred because it is JVM vendor-independent. Java Broker only.

Table 10.5. List of ACL rules

UserManagement	User maintainance; create/delete/view users, change passwords etc	permissionable at broker level only
ConfigurationManagement	Dynammically reload configuration from disk.	permissionable at broker level only
LoggingManagement	Dynammically control Qpid logging level	permissionable at broker level only
ServerInformation	Read-only information regarding the Qpid: version number etc	permissionable at broker level only
VirtualHost.Queue	Queue maintainance; copy/move/purge/view etc	
VirtualHost.Exchange	Exchange maintenance; bind/unbind queues to exchanges	
VirtualHost.VirtualHost	Virtual host maintainace; create/delete exchanges, queues etc	

10.4.4. Worked Examples

Here are some example ACLs illustrating common use cases. In addition, note that the Java broker provides a complete example ACL file, located at etc/broker_example.acl.

10.4.4.1. Worked example 1 - Management rights

Suppose you wish to permission two users: a user 'operator' must be able to perform all Management operations, and a user 'readonly' must be enable to perform only read-only functions. Neither 'operator' nor 'readonly' should be allowed to connect clients for messaging.

```
# Deny (loggged) operator/readonly permission to connect messaging clients.
ACL DENY-LOG operator ACCESS VIRTUALHOST
ACL DENY-LOG readonly ACCESS VIRTUALHOST
# Give operator permission to perfom all other actions
ACL ALLOW operator ALL ALL
# Give readonly permission to execute only read-only actions
ACL ALLOW readonly ACCESS ALL
...
... rules for other users
...
# Explicitly deny all (log) to eveyone
ACL DENY-LOG ALL ALL
```

10.4.4.2. Worked example 2 - User maintainer group

Suppose you wish to restrict User Management operations to users belonging to a group 'usermaint'. No other user is allowed to perform user maintainence This example illustrates the permissioning of an individual component.

```
# Give usermaint access to management and permission to execute all JMX Methods on
# UserManagement MBean and perform all actions for USER objects
ACL ALLOW usermaint ACCESS MANAGEMENT
ACL ALLOW usermaint ALL METHOD component="UserManagement"
ACL ALLOW usermaint ALL USER
ACL DENY ALL ALL METHOD component="UserManagement"
ACL DENY ALL ALL USER
...
... rules for other users
...
ACL DENY-LOG ALL ALL
```

10.4.4.3. Worked example 3 - Request/Response messaging

Suppose you wish to permission a system using a request/response paradigm. Two users: 'client' publishes requests; 'server' consumes the requests and generates a response. This example illustrates the permissioning of AMQP exchanges and queues.

```
# Allow client and server to connect to the virtual host.
ACL ALLOW client ACCESS VIRTUALHOST
```

```
ACL ALLOW server ACCESS VIRTUALHOST

# Client side
# Allow the 'client' user to publish requests to the request queue. As is the norm
# is required to create a temporary queue on which the server will respond.  Conse
# of the temporary queues and consumption of messages from it.
ACL ALLOW client CREATE QUEUE temporary="true"
ACL ALLOW client CONSUME QUEUE temporary="true"
ACL ALLOW client DELETE QUEUE temporary="true"
ACL ALLOW client BIND EXCHANGE name="amq.direct" temporary="true"
ACL ALLOW client UNBIND EXCHANGE name="amq.direct" temporary="true"
ACL ALLOW client PUBLISH EXCHANGE name="amq.direct" routingKey="example.RequestQue

# Server side
# Allow the 'server' user to consume from the request queue and publish a response
# client.  We also allow the server to create the request queue.
ACL ALLOW server CREATE QUEUE name="example.RequestQueue"
ACL ALLOW server CONSUME QUEUE name="example.RequestQueue"
ACL ALLOW server BIND EXCHANGE
ACL ALLOW server PUBLISH EXCHANGE name="amq.direct" routingKey="TempQueue*"

ACL DENY-LOG all all
```

10.4.4.4. Worked example 4 - firewall-like access control

This example illustrates how to set up an ACL that restricts the IP addresses and hostnames of messaging clients that can access a virtual host.

```
################
# Hostname rules
################

# Allow messaging clients from company1.com and company1.co.uk to connect
ACL ALLOW all ACCESS VIRTUALHOST from_hostname=".*\.company1\.com,.*\.company1\.co

# Deny messaging clients from hosts within the dev subdomain
ACL DENY-LOG all ACCESS VIRTUALHOST from_hostname=".*\.dev\.company1\.com"

##################
# IP address rules
##################

# Deny access to all users in the IP ranges 192.168.1.0-192.168.1.255 and 192.168.
# using the notation specified in RFC 4632, "Classless Inter-domain Routing (CIDR)
ACL DENY-LOG messaging-users ACCESS VIRTUALHOST \
  from_network="192.168.1.0/24,192.168.2.0/24"

# Deny access to all users in the IP ranges 192.169.1.0-192.169.1.255 and 192.169.
# using wildcard notation.
ACL DENY-LOG messaging-users ACCESS VIRTUALHOST \
  from_network="192.169.1.*,192.169.2.*"
```

```
ACL DENY-LOG all all
```

10.5. SSL

This section will show how to use SSL to enable secure connections between an AMQP message client and the broker.

10.5.1. Keystore Configuration

The broker configuration file (config.xml) needs to be updated to include the required SSL keystore configuration, an example of which can be found below.

Example 10.6. Configuring an SSL Keystore

```
<connector>
  ...
  <ssl>
    <enabled>true</enabled>
    <port>5671</port>
    <sslOnly>false</sslOnly>
    <keyStorePath>/path/to/keystore.ks</keyStorePath>
    <keyStorePassword>keystorepass</keyStorePassword>
    <certAlias>alias<certAlias>
  </ssl>
  ...
<connector>
```

The certAlias element is an optional way of specifying which certificate the broker should use if the keystore contains multiple entries.

The sslOnly element controls whether the broker will **only** bind the configured SSL port(s) or will also bind the non-SSL port(s). Setting sslOnly to true will disable the non-SSL ports.

Important

The password of the certificate used by the Broker **must** match the password of the keystore itself. This is a restriction of the Qpid Broker implementation. If using the keytool [http://docs.oracle.com/javase/6/docs/technotes/tools/solaris/keytool.html] utility, note that this means the argument to the `-keypass` option must match the `-storepass` option.

10.5.2. Truststore / Client Certificate Authentication

The SSL trustore and related Client Certificate Authentication behaviour can be configured with additional configuration as shown in the example below, in which the broker requires client certificate authentication.

Example 10.7. Configuring an SSL Truststore and client auth

```
<connector>
  ...
  <ssl>
    ...
    <trustStorePath>/path/to/truststore.ks</trustStorePath>
    <trustStorePassword>truststorepass</trustStorePassword>
    <needClientAuth>true</needClientAuth>
    <wantClientAuth>false</wantClientAuth>
    ...
  </ssl>
  ...
<connector>
```

The needClientAuth and wantClientAuth elements allow control of whether the client must present an SSL certificate. Only one of these elements is needed but both may be used at the same time. A socket's client authentication setting is one of three states: required (needClientAuth = true), requested (wantClientAuth = true), or none desired (both false, the default). If both elements are set to true, needClientAuth takes precedence.

When using Client Certificate Authentication it may be desirable to use the External Authentication Manager, for details see Section 10.3.4, “External (SSL Client Certificates)”

Chapter 11. Runtime

11.1. Log Files

11.2. Alerts

11.3. Disk Space Management

11.3.1. Producer Flow Control

11.3.1.1. General Information

The Qpid 0.6 release introduced a simplistic producer-side flow control mechanism into the Java Messaging Broker, causing producers to be flow-controlled when they attempt to send messages to an overfull queue. Qpid 0.18 introduced a similar mechanism triggered by an overfull persistent message store on a virtual host.

11.3.1.2. Server Configuration

11.3.1.2.1. Configuring a Queue to use flow control

Flow control is enabled on a producer when it sends a message to a Queue which is "overfull". The producer flow control will be rescinded when all Queues on which a producer is blocking become "underfull". A Queue is defined as overfull when the size (in bytes) of the messages on the queue exceeds the "capacity" of the Queue. A Queue becomes "underfull" when its size becomes less than the "flowResumeCapacity".

Example 11.1. Configuring a queue depth limit

```
<queue>
    <name>test</name>
    <test>
        <exchange>amq.direct</exchange>
        <capacity>10485760</capacity>                    <!-- set the queue capac
        <flowResumeCapacity>8388608</flowResumeCapacity>  <!-- set the resume capa
    </test>
</queue>
```

The default for all queues on a virtual host can also be set

Example 11.2. Configuring a default queue depth limit on a virtualhost

```
<virtualhosts>
    <virtualhost>
        <name>localhost</name>
        <localhost>
            <capacity>10485760</capacity>                    <!-- set the queue c
            <flowResumeCapacity>8388608</flowResumeCapacity>  <!-- set the resume
        </localhost>
    </virtualhost>
</virtualhosts>
```

Where no flowResumeCapacity is set, the flowResumeCapacity is set to be equal to the capacity. Where no capacity is set, capacity is defaulted to 0 meaning there is no capacity limit.

11.3.1.2.1.1. Broker Log Messages

There are four new Broker log messages that may occur if flow control through queue capacity limits is enabled. Firstly, when a capacity limited queue becomes overfull, a log message similar to the following is produced

```
MESSAGE [vh(/test)/qu(MyQueue)] [vh(/test)/qu(MyQueue)] QUE-1003 : Overfull : Size
```

Then for each channel which becomes blocked upon the overful queue a log message similar to the following is produced:

```
MESSAGE [con:2(guest@anonymous(713889609)/test)/ch:1] [con:2(guest@anonymous(71388
```

When enough messages have been consumed from the queue that it becomes underfull, then the following log is generated:

```
MESSAGE [vh(/test)/qu(MyQueue)] [vh(/test)/qu(MyQueue)] QUE-1004 : Underfull : Siz
```

And for every channel which becomes unblocked you will see a message similar to:

```
MESSAGE [con:2(guest@anonymous(713889609)/test)/ch:1] [con:2(guest@anonymous(71388
```

Obviously the details of connection, virtual host, queue, size, capacity, etc would depend on the configuration in use.

11.3.1.2.2. Disk quota-based flow control

Since version 0.18 of Qpid Broker, flow control can be triggered when a configured disk quota is exceeded. This is supported by the BDB and Derby message stores.

This functionality blocks all producers on reaching the disk overflow limit. When consumers consume the messages, causing disk space usage to falls below the underflow limit, the producers are unblocked and continue working as normal.

Two limits can be configured:

overfull limit - the maximum space on disk (in bytes) which can be used by store.

underfull limit - when the space on disk drops below this limit, producers are allowed to resume publishing.

An example of quota configuration for the BDB message store is provided below.

Example 11.3. Configuring a limit on a store

```
<store>
   <class>org.apache.qpid.server.store.berkeleydb.BDBMessageStore</class>
   <environment-path>${work}/bdbstore/test</environment-path>
   <overfull-size>50000000</overfull-size>
   <underfull-size>45000000</underfull-size>
</store>
```

The disk quota functionality is based on "best effort" principle. This means the broker cannot guarantee that the disk space limit will not be exceeded. If several concurrent transactions are started before the limit is reached, which collectively cause the limit to be exceeded, the broker may allow all of them to be committed.

11.3.1.2.2.1. Broker Log Messages for quota flow control

There are 2 new broker log messages that may occur if flow control through disk quota limits is enabled. When the virtual host is blocked due to exceeding of the disk quota limit the following message appears in the broker log

```
[vh(/test)/ms(BDBMessageStore)] MST-1008 : Store overfull, flow control will be en
```

When virtual host is unblocked after cleaning the disk space the following message appears in the broker log

```
[vh(/test)/ms(BDBMessageStore)] MST-1009 : Store overfull condition cleared
```

11.3.1.3. Client impact and configuration

If a producer sends to a queue which is overfull, the broker will respond by instructing the client not to send any more messages. The impact of this is that any future attempts to send will block until the broker rescinds the flow control order.

While blocking the client will periodically log the fact that it is blocked waiting on flow control.

```
WARN   Message send delayed by 5s due to broker enforced flow control
WARN   Message send delayed by 10s due to broker enforced flow control
```

After a set period the send will timeout and throw a JMSException to the calling code.

If such a JMSException is thrown, the message will not be sent to the broker, however the underlying Session may still be active - in particular if the Session is transactional then the current transaction will not be automatically rolled back. Users may choose to either attempt to resend the message, or to roll back any transactional work and close the Session.

Both the timeout delay and the periodicity of the warning messages can be set using Java system properties.

The amount of time (in milliseconds) to wait before timing out is controlled by the property qpid.flow_control_wait_failure.

The frequency at which the log message informing that the producer is flow controlled is sent is controlled by the system property qpid.flow_control_wait_notify_period.

Adding the following to the command line to start the client would result in a timeout of one minute, with warning messages every ten seconds:

```
-Dqpid.flow_control_wait_failure=60000
-Dqpid.flow_control_wait_notify_period=10000
```

11.3.1.3.1. Older Clients

The flow control feature was first added to the Java broker/client in the 0.6 release. If an older client connects to the broker then the flow control commands will be ignored by it and it will not be blocked. So to fully benefit from this feature both Client and Broker need to be at least version 0.6.

11.4. Producer Transaction Timeout

11.4.1. General Information

The transaction timeout mechanism is used to control broker resources when clients producing messages using transactional sessions hang or otherwise become unresponsive, or simply begin a transaction and keep using it without ever calling Session#commit() [http://docs.oracle.com/javaee/6/api/javax/jms/Session.html#commit].

Users can choose to configure an idleWarn or openWarn threshold, after which the identified transaction should be logged as a WARN level alert as well as (more importantly) an idleClose or openClose threshold after which the transaction and the connection it applies to will be closed.

This feature is particularly useful in environments where the owner of the broker does not have full control over the implementation of clients, such as in a shared services deployment.

The following section provide more details on this feature and its use.

11.4.2. Purpose

This feature has been introduced to address the scenario where an open transaction on the broker holds an open transaction on the persistent store. This can have undesirable consequences if the store does not time

out or close long-running transactions, such as with BDB. This can can result in a rapid increase in disk usage size, bounded only by available space, due to growth of the transaction log.

11.4.3. Scope

Note that only MessageProducer [http://docs.oracle.com/javaee/6/api/javax/jms/MessageProducer.html] clients will be affected by a transaction timeout, since store transaction lifespan on a consumer only spans the execution of the call to Session#commit() and there is no scope for a long-lived transaction to arise.

It is also important to note that the transaction timeout mechanism is purely a JMS transaction timeout, and unrelated to any other timeouts in the Qpid client library and will have no impact on any RDBMS your application may utilise.

11.4.4. Effect

Full details of configuration options are provided in the sections that follow. This section gives a brief overview of what the Transaction Timeout feature can do.

11.4.4.1. Broker Logging and Connection Close

When the openWarn or idleWarn specified threshold is exceeded, the broker will log a WARN level alert with details of the connection and channel on which the threshold has been exceeded, along with the age of the transaction.

When the openClose or idleClose specified threshold value is exceeded, the broker will throw an exception back to the client connection via the ExceptionListener [http://docs.oracle.com/javaee/6/api/javax/jms/ExceptionListener.html], log the action and then close the connection.

The example broker log output shown below is where the idleWarn threshold specified is lower than the idleClose threshold and the broker therefore logs the idle transaction 3 times before the close threshold is triggered and the connection closed out.

```
CHN-1008 : Idle Transaction : 13,116 ms
CHN-1008 : Idle Transaction : 14,116 ms
CHN-1008 : Idle Transaction : 15,118 ms
CHN-1003 : Close
```

The second example broker log output shown below illustrates the same mechanism operating on an open transaction.

```
CHN-1007 : Open Transaction : 12,406 ms
CHN-1007 : Open Transaction : 13,406 ms
CHN-1007 : Open Transaction : 14,406 ms
CHN-1003 : Close
```

11.4.4.2. Client Side Effect

After a Close threshold has been exceeded, the trigger client will receive this exception on its exception listener [http://docs.oracle.com/javaee/6/api/javax/jms/ExceptionListener.html], prior to being disconnected:

```
org.apache.qpid.AMQConnectionClosedException: Error: Idle transaction
timed out [error code 506: resource error]
```

Any later attempt to use the connection will result in this exception being thrown:

```
Producer: Caught an Exception: javax.jms.IllegalStateException: Object org.apache.
    javax.jms.IllegalStateException: Object org.apache.qpid.client.AMQSession_0_8@
    at org.apache.qpid.client.Closeable.checkNotClosed(Closeable.java:70)
    at org.apache.qpid.client.AMQSession.checkNotClosed(AMQSession.java:555)
    at org.apache.qpid.client.AMQSession.createBytesMessage(AMQSession.java:573)
```

Thus clients must be able to handle this case successfully, reconnecting where required and registering an exception listener on all connections. This is critical, and must be communicated to client applications by any broker owner switching on transaction timeouts.

11.4.5. Configuration

11.4.5.1. Configuration

Transaction timeouts are configurable separately on each defined virtual host, using the virtualhosts.xml file.

We would recommend that only warnings are configured at first, which should allow broker administrators to obtain an idea of the distribution of transaction lengths on their systems, and configure production settings appropriately for both warning and closure. Ideally establishing thresholds should be achieved in a representative UAT environment, with clients and broker running, prior to any production deployment.

It is impossible to give suggested values, due to the large variation in usage depending on the applications using a broker. However, clearly transactions should not span the expected lifetime of any client application as this would indicate a hung client.

When configuring warning and closure timeouts, it should be noted that these only apply to message producers that are connected to the broker, but that a timeout will cause the connection to be closed - this disconnecting all producers and consumers created on that connection.

This should not be an issue for environments using Mule or Spring, where connection factories can be configured appropriately to manage a single MessageProducer object per JMS Session and Connection. Clients that use the JMS API directly should be aware that sessions managing both consumers and producers, or multiple producers, will be affected by a single producer hanging or leaving a transaction idle or open, and closed, and must take appropriate action to handle that scenario.

11.4.5.2. Virtualhosts.xml

The JMS transaction timeouts are configured on each virtual host defined in the XML configuration files.

The default values for each of the parameters is 0, indicating that the particular check is disabled.

Any or all of the parameters can be set, using the desired value in milliseconds, and will be checked each time the housekeeping process runs, usually set to run every 30 seconds in standard configuration. The meaning of each property is as follows:

- openWarn - the time a transaction can be open for (with activity occurring on it) after which a warning alert will be issued.

- openClose - the time a transaction can be open for before the connection it is on is closed.
- idleWarn - the time a transaction can be idle for (with no activity occurring on it) after which a warning alert will be issued.
- idleClose - the time a transaction can be idle for before the connection it is on is closed.

The virtualhosts configuration is shown below, and must occur inside the //virtualhosts/virtualhost/name/ elements:

Example 11.4. Configuring producer transaction timeout

```
<transactionTimeout>
    <openWarn>10000</openWarn>
    <openClose>20000</openClose>
    <idleWarn>5000</idleWarn>
    <idleClose>15000</idleClose>
</transactionTimeout>
```

11.5. Handing Undeliverable Messages

11.5.1. Introduction

Messages that cannot be delivered successfully to a consumer (for instance, because the client is using a transacted session and rolls-back the transaction) can be made available on the queue again and then subsequently be redelivered, depending on the precise session acknowledgement mode and messaging model used by the application. This is normally desirable behaviour that contributes to the ability of a system to withstand unexpected errors. However, it leaves open the possibility for a message to be repeatedly redelivered (potentially indefinitely), consuming system resources and preventing the delivery of other messages. Such undeliverable messages are sometimes known as poison messages.

For an example, consider a stock ticker application that has been designed to consume prices contained within JMS TextMessages. What if inadvertently a BytesMessage is placed onto the queue? As the ticker application does not expect the BytesMessage, its processing might fail and cause it to roll-back the transaction, however the default behavior of the Broker would mean that the BytesMessage would be delivered over and over again, preventing the delivery of other legitimate messages, until an operator intervenes and removes the erroneous message from the queue.

Qpid has maximum delivery count and dead-letter queue (DLQ) features which can be used in concert to construct a system that automatically handles such a condition. These features are described in the following sections.

11.5.2. Maximum Delivery Count

Maximum delivery count is a property of a queue. If a consumer application is unable to process a message more than the specified number of times, then the broker will either route the message to a dead-letter queue (if one has been defined), or will discard the message.

In order for a maximum delivery count to be enforced, the consuming client *must* call Session#rollback() [http://docs.oracle.com/javaee/6/api/javax/jms/Session.html#rollback()] (or Session#recover() [http://docs.oracle.com/javaee/6/api/javax/jms/Session.html#recover()] if the session is not transacted). It is

during the Broker's processing of Session#rollback() (or Session#recover()) that if a message has been seen at least the maximum number of times then it will move the message to the DLQ or discard the message.

If the consuming client fails in another manner, for instance, closes the connection, the message will not be re-routed and consumer application will see the same poison message again once it reconnects.

If the consuming application is using AMQP 0-9-1, 0-9, or 0-8 protocols, it is necessary to set the client system property `qpid.reject.behaviour` or connection or binding URL option `rejectbehaviour` to the value `system`.

It is possible to determine the number of times a message has been sent to a consumer via the Management interfaces, but is not possible to determine this information from a message client. Specifically, the optional JMS message header JMSXDeliveryCount is not supported.

Maximum Delivery Count can be enabled via management (see Chapter 9, *Configuring And Managing*) using the the queue declare property x-qpid-maximum-delivery-count or via configuration as illustrated below.

11.5.3. Dead Letter Queues (DLQ)

A Dead Letter Queue (DLQ) acts as an destination for messages that have somehow exceeded the normal bounds of processing and is utilised to prevent disruption to flow of other messages. When a DLQ is enabled for a given queue if a consuming client indicates it no longer wishes the receive the message (typically by exceeding a Maximum Delivery Count) then the message is moved onto the DLQ and removed from the original queue.

The DLQ feature causes generation of a Dead Letter Exchange and a Dead Letter Queue. These are named convention QueueName*_DLE* and QueueName*_DLQ*.

DLQs can be enabled via management (see Chapter 9, *Configuring And Managing*) using the queue declare property x-qpid-dlq-enabled or via configuration as illustrated below.

Avoid excessive queue depth

Applications making use of DLQs *should* make provision for the frequent examination of messages arriving on DLQs so that both corrective actions can be taken to resolve the underlying cause and organise for their timely removal from the DLQ. Messages on DLQs consume system resources in the same manner as messages on normal queues so excessive queue depths should not be permitted to develop.

11.5.4. Configuration

In the below configuration it can be seen that DLQs/Maximum Delivery Count are enabled at the broker level with maximum delivery count set to 5, disabled at the virtualhost level for the 'dev-only' virtualhost, and enabled specifically for the 'dev-only-main-queue' with maximum delivery count overridden to 5.

As 'dev-only-main-queue' has its own configuration specified, this value overrides all others and causes the features to be enabled for this queue. In contrast to this, 'dev-only-other-queue' does not specify its own value and picks up the false value specified for its parent virtualhost, causing the DLQ/Maximum Delivery Count features to be disabled for this queue. Any such queue in the 'dev-only' virtualhost which does not specify its own configuration value will have the DLQ/Maximum Delivery Count feature disabled.

The queue 'localhost-queue' has the DLQ/Maximum Delivery Count features enabled, as neither the queue itself or the 'localhost' virtualhost specifies a configuration value and so the broker level value of true is

used. Any such queue in the 'localhost' virtualhost which does not specify its own configuration value will have the features enabled.

Example 11.5. Enabling DLQs and maximum delivery count at broker level within config.xml

```
<broker>
 ...
 <deadLetterQueues>true</deadLetterQueues>
 <maximumDeliveryCount>5</maximumDeliveryCount>
 ...
</broker>
```

Example 11.6. Enabling DLQs and maximum delivery count at virtualhost and queue level within virtualhosts.xml

```
<virtualhosts>
 ...
 <virtualhost>
  <name>dev-only</name>
  <dev-only>
   <queues>
    <deadLetterQueues>false</deadLetterQueues>
    <maximumDeliveryCount>0</maximumDeliveryCount>
    <queue>
     <name>dev-only-main-queue</name>
     <dev-only-main-queue>
      <deadLetterQueues>true</deadLetterQueues>
      <maximumDeliveryCount>3</maximumDeliveryCount>
     </dev-only-main-queue>
    </queue>
    <queue>
     <name>dev-only-other-queue</name>
    </queue>
   </queues>
  </dev-only>
 </virtualhost>
 <virtualhost>
  <name>localhost</name>
  <localhost>
   <queues>
    <queue>
     <name>localhost-queue</name>
    </queue>
   </queues>
  </localhost>
 </virtualhost>
 ...
</virtualhosts>
```

Chapter 12. High Availability

12.1. General Introduction

The term High Availability (HA) usually refers to having a number of instances of a service such as a Message Broker available so that should a service unexpectedly fail, or requires to be shutdown for maintenance, users may quickly connect to another instance and continue their work with minimal interuption. HA is one way to make a overall system more resilient by eliminating a single point of failure from a system.

HA offerings are usually categorised as **Active/Active** or **Active/Passive**. An Active/Active system is one where all nodes within the cluster are usuaully available for use by clients all of the time. In an Active/Passive system, one only node within the cluster is available for use by clients at any one time, whilst the others are in some kind of standby state, awaiting to quickly step-in in the event the active node becomes unavailable.

12.2. HA offerings of the Java Broker

The Java Broker's HA offering became available at release **0.18**. HA is provided by way of the HA features built into the Java Edition of the Berkley Database (BDB JE) [http://www.oracle.com/technetwork/products/berkeleydb/overview/index-093405.html] and as such is currently only available to Java Broker users who use the optional BDB JE based persistence store. This **optional** store requires the use of BDB JE which is licensed under the Sleepycat Licence, which is not compatible with the Apache Licence and thus BDB JE is not distributed with Qpid. Users who elect to use this optional store for the broker have to provide this dependency.

HA in the Java Broker provides an **Active/Passive** mode of operation with Virtual hosts being the unit of replication. The Active node (referred to as the **Master**) accepts all work from all the clients. The Passive nodes (referred to as **Replicas**) are unavailable for work: the only task they must perform is to remain in synch with the Master node by consuming a replication stream containing all data and state.

If the Master node fails, a Replica node is elected to become the new Master node. All clients automatically failover [1] to the new Master and continue their work.

The Java Broker HA solution is incompatible with the HA solution offered by the CPP Broker. It is not possible to co-locate Java and CPP Brokers within the same cluster.

HA is not currently available for those using the the **Derby Store** or **Memory Message Store**.

12.3. Two Node Cluster

12.3.1. Overview

In this HA solution, a cluster is formed with two nodes. one node serves as **master** and the other is a **replica**.

All data and state required for the operation of the virtual host is automatically sent from the master to the replica. This is called the replication stream. The master virtual host confirms each message is on the

[1] The automatic failover feature is available only for AMQP connections from the Java client. Management connections (JMX) do not current offer this feature.

replica before the client transaction completes. The exact way the client awaits for the master and replica is gorverned by the durability configuration, which is discussed later. In this way, the replica remains ready to take over the role of the master if the master becomes unavailable.

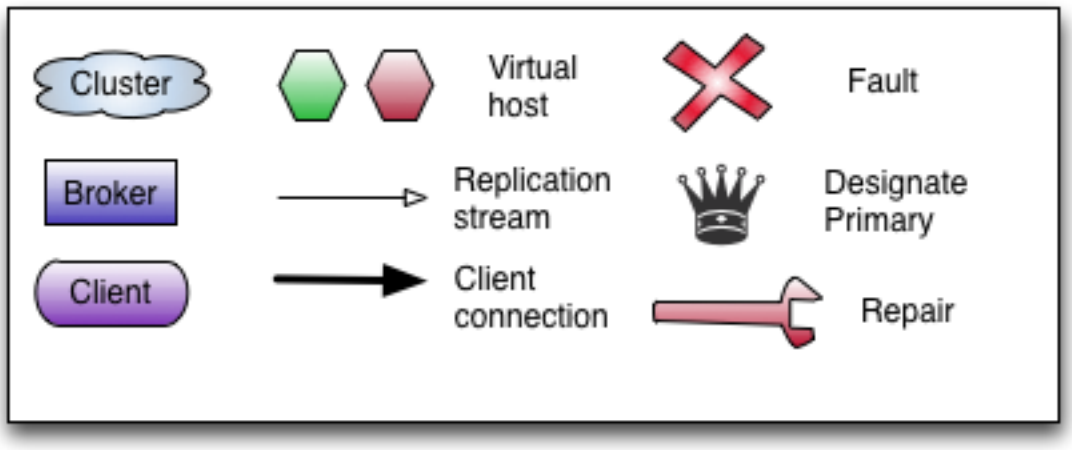
It is important to note that there is an inherent limitation of two node clusters is that the replica node cannot make itself master automatically in the event of master failure. This is because the replica has no way to distinguish between a network partition (with potentially the master still alive on the other side of the partition) and the case of genuine master failure. (If the replica were to elect itself as master, the cluster would run the risk of a split-brain [http://en.wikipedia.org/wiki/Split-brain_(computing)] scenario). In the event of a master failure, a third party must designate the replica as primary. This process is described in more detail later.

Clients connect to the cluster using a failover url. This allows the client to maintain a connection to the master in a way that is transparent to the client application.

12.3.2. Depictions of cluster operation

In this section, the operation of the cluster is depicted through a series of figures supported by explanatory text.

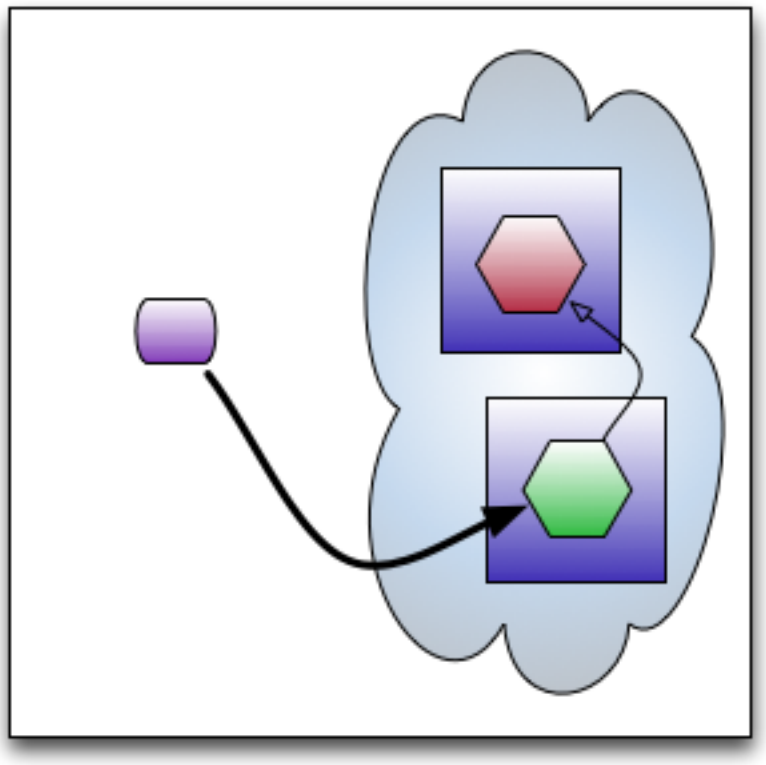
Figure 12.1. Key for figures



12.3.2.1. Normal Operation

The figure below illustrates normal operation. Clients connecting to the cluster by way of the failover URL achieve a connection to the master. As clients perform work (message production, consumption, queue creation etc), the master additionally sends this data to the replica over the network.

Figure 12.2. Normal operation of a two-node cluster



12.3.2.2. Master Failure and Recovery

The figure below illustrates a sequence of events whereby the master suffers a failure and the replica is made the master to allow the clients to continue to work. Later the old master is repaired and comes back on-line in replica role.

The item numbers in this list apply to the numbered boxes in the figure below.

1. System operating normally

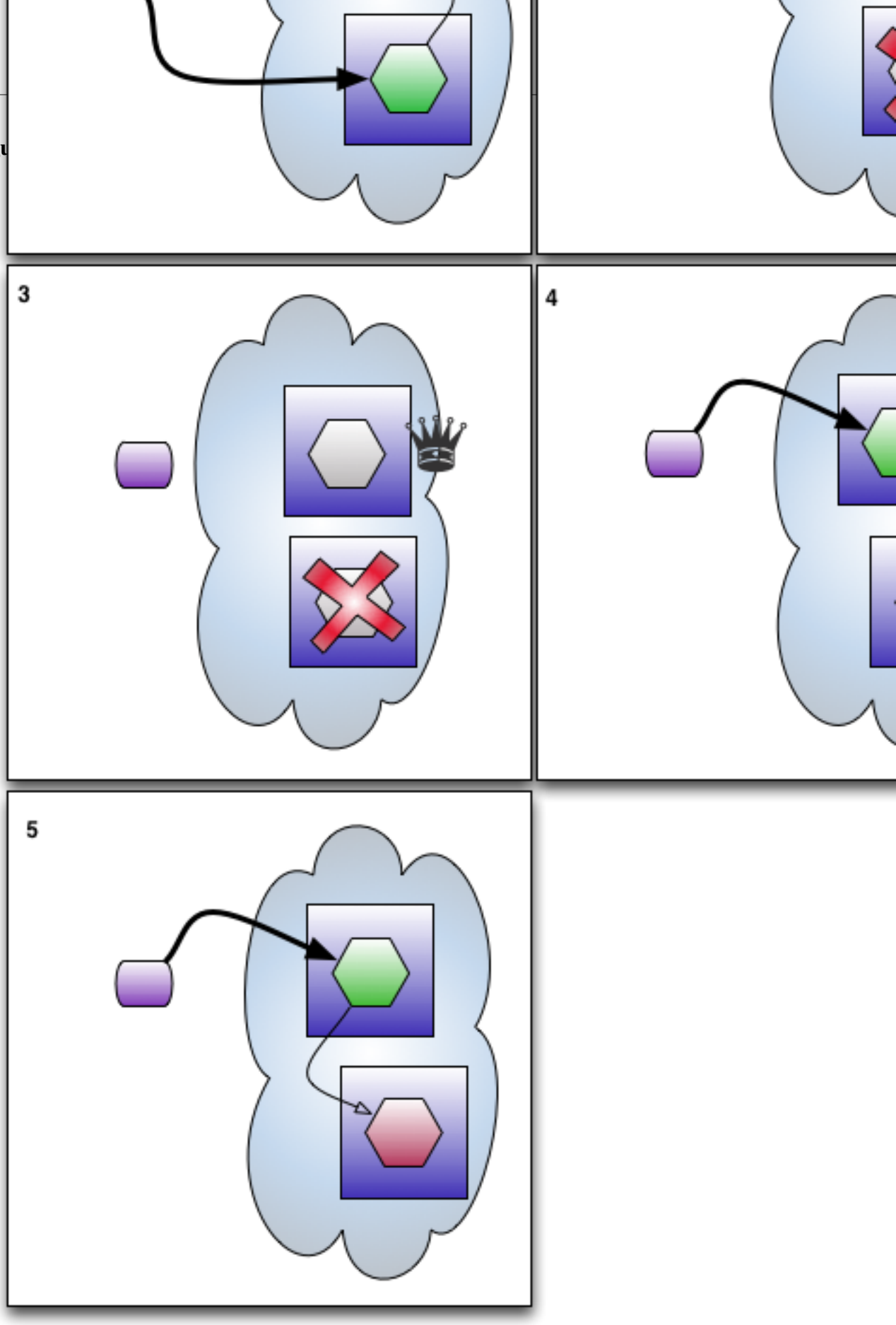
2. Master suffers a failure and disconnects all clients. Replica realises that it is no longer in contact with master. Clients begin to try to reconnect to the cluster, although these connection attempts will fail at this point.

3. A third-party (an operator, a script or a combination of the two) verifies that the master has truely failed **and is no longer running**. If it has truely failed, the decision is made to designate the replica as primary, allowing it to assume the role of master despite the other node being down. This primary designation is performed using JMX.

4. Client connections to the new master succeed and the **service is restored** , albeit without a replica.

5. The old master is repaired and brought back on-line. It automatically rejoins the cluster in the **replica** role.

Figu



12.3.2.3. Replica Failure and Recovery

The figure that follows illustrates a sequence of events whereby the replica suffers a failure leaving the master to continue processing alone. Later the replica is repaired and is restarted. It rejoins the cluster so that it is once again ready to take over in the event of master failure.

The behavior of the replica failure case is governed by the `designatedPrimary` configuration item. If set true on the master, the master will continue to operate solo without outside intervention when the replica fails. If false, a third-party must designate the master as primary in order for it to continue solo.

The item numbers in this list apply to the numbered boxes in the figure below. This example assumes that `designatedPrimary` is true on the original master node.

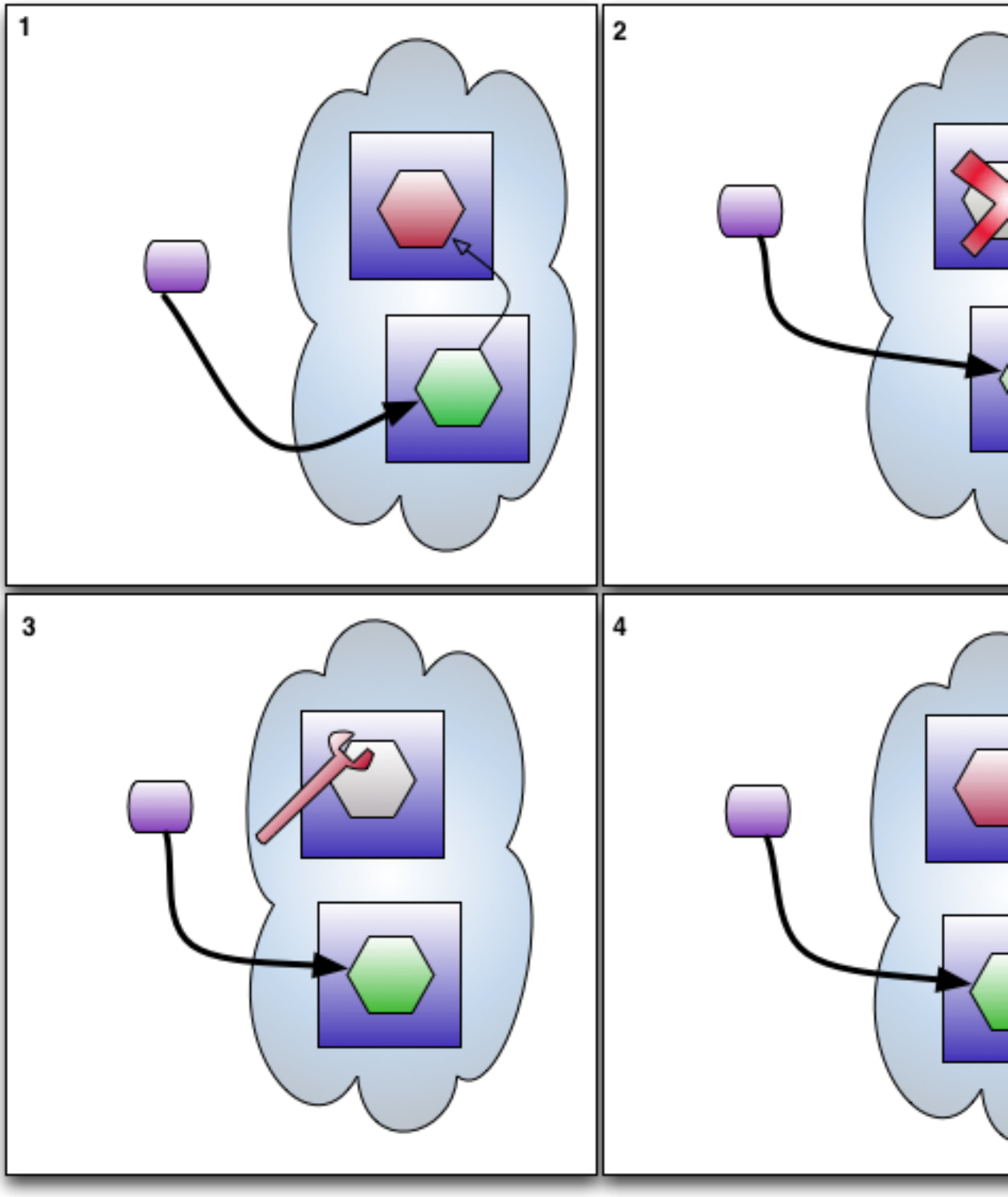
1. System operating normally

2. Replica suffers a failure. Master realises that replica longer in contact but as `designatedPrimary` is true, master continues processing solo and thus client connections are uninterrupted by the loss of the replica. System continues operating normally, albeit with a single node.

3. Replica is repaired.

4. After catching up with missed work, replica is once again ready to take over in the event of master failure.

Figure 12.4. Failure of replica and subsequent recovery sequence



12.3.2.4. Network Partition and Recovery

The figure below illustrates the sequence of events that would occur if the network between master and replica were to suffer a partition, and the nodes were out of contact with one and other.

As with Replica Failure and Recovery, the behaviour is governed by the `designatedPrimary`. Only if `designatedPrimary` is true on the master, will the master continue solo.

The item numbers in this list apply to the numbered boxes in the figure below. This example assumes that `designatedPrimary` is true on the original master node.

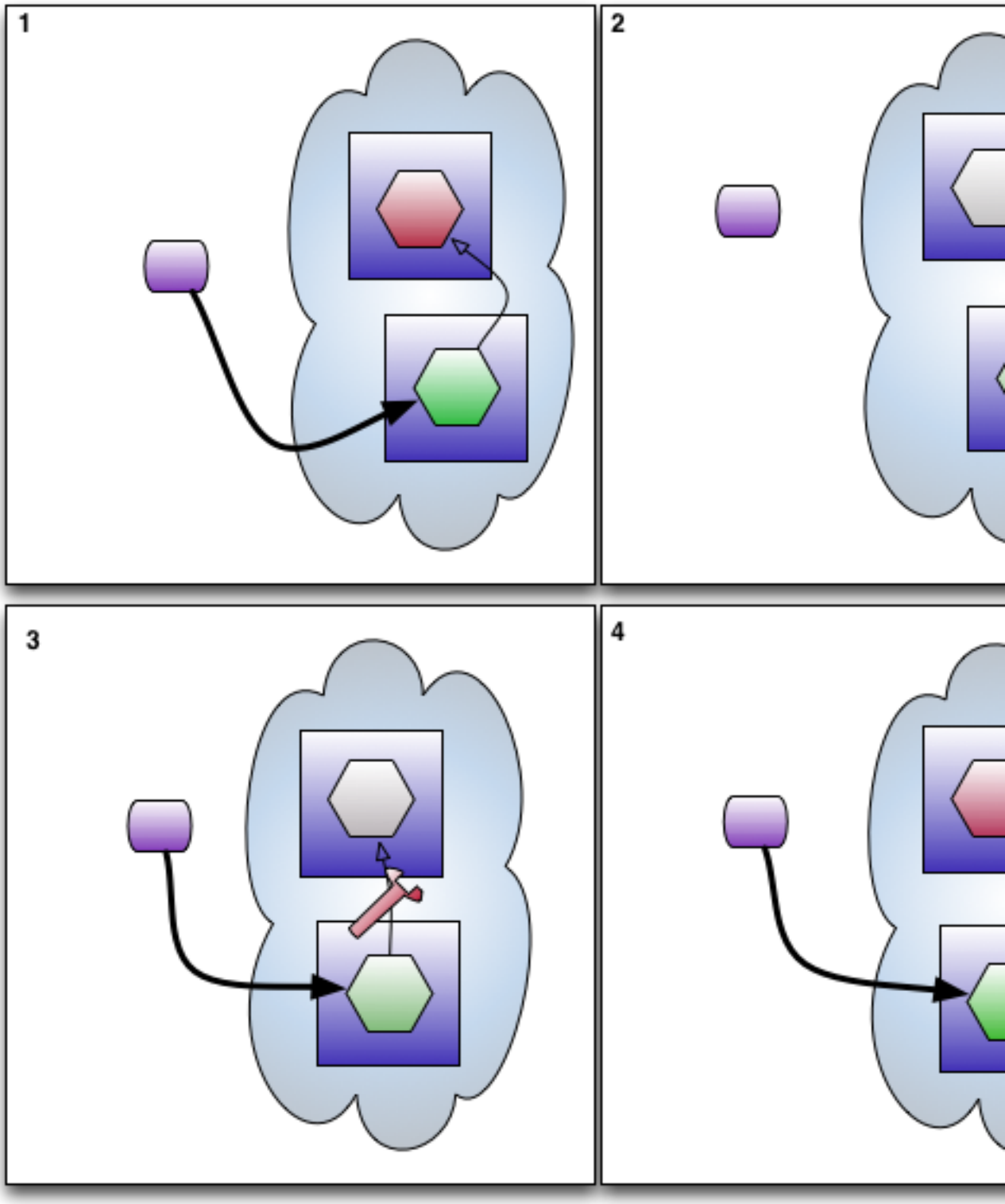
1. System operating normally

2. Network suffers a failure. Master realises that replica longer in contact but as `designatedPrimary` is true, master continues processing solo and thus client connections are uninterrupted by the network partition between master and replica.

3. Network is repaired.

4. After catching up with missed work, replica is once again ready to take over in the event of master failure. System operating normally again.

Figure 12.5. Partition of the network separating master and replica



12.3.2.5. Split Brain

A split-brain [http://en.wikipedia.org/wiki/Split-brain_(computing)] is a situation where the two node cluster has two masters. BDB normally strives to prevent this situation arising by preventing two nodes in a cluster being master at the same time. However, if the network suffers a partition, and the third-party intervenes incorrectly and makes the replica a second master a split-brain will be formed and both masters will proceed to perform work **independently** of one and other.

There is no automatic recovery from a split-brain.

Manual intervention will be required to choose which store will be retained as master and which will be discarded. Manual intervention will be required to identify and repeat the lost business transactions.

The item numbers in this list apply to the numbered boxes in the figure below.

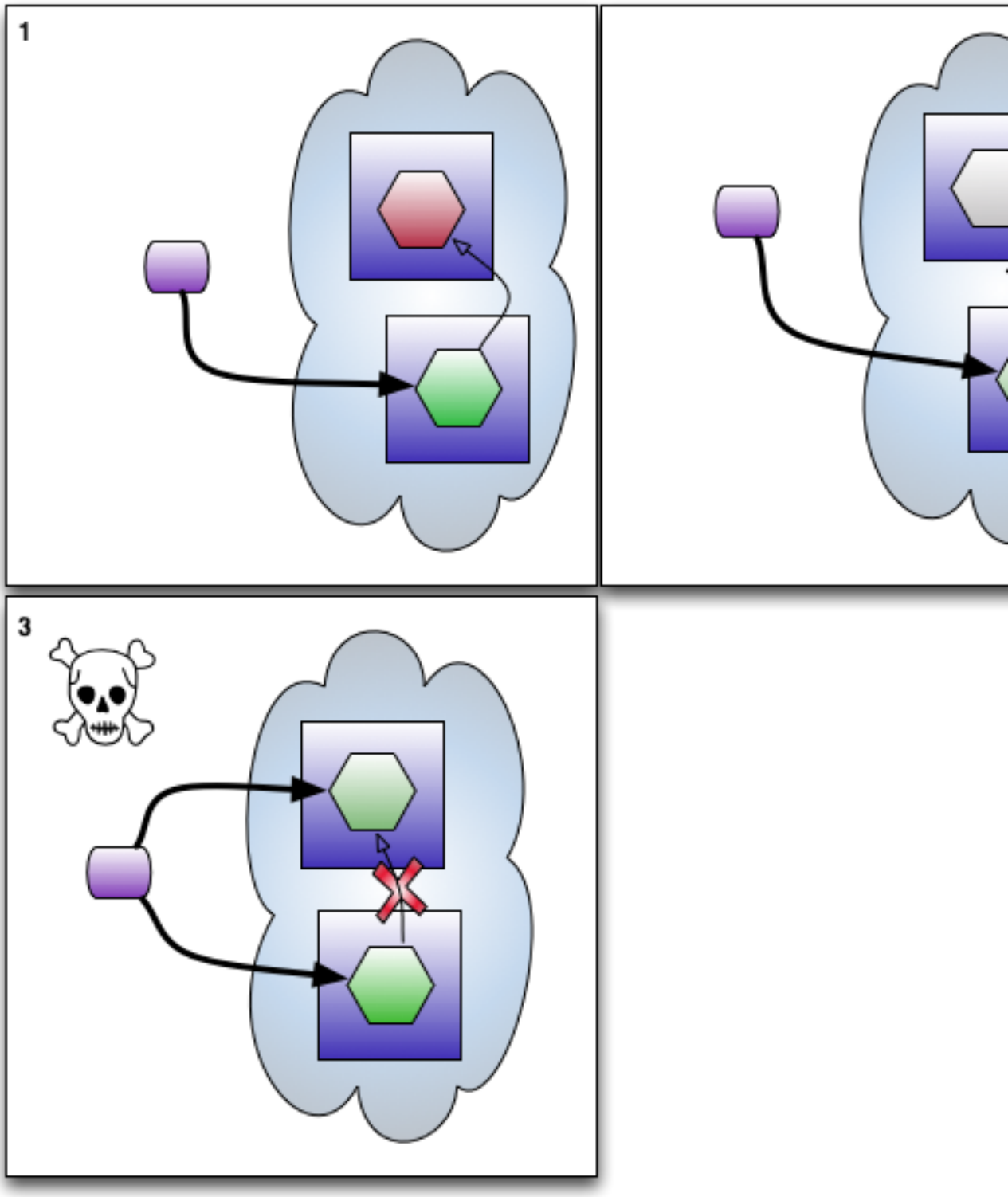
1. System operating normally

2. Network suffers a failure. Master realises that replica longer in contact but as `designatedPrimary` is true, master continues processing solo. Client connections are uninterrupted by the network partition.

   A third-party **erroneously** designates the replica as primary while the original master continues running (now solo).

3. As the nodes cannot see one and other, both behave as masters. Clients may perform work against both master nodes.

Figure 12.6. Split Brain



12.4. Multi Node Cluster

Multi node clusters, that is clusters where the number of nodes is three or more, are not yet ready for use.

12.5. Configuring a Virtual Host to be a node

To configure a virtualhost as a cluster node, configure the virtualhost.xml in the following manner:

Example 12.1. Configuring a VirtualHost to use the BDBHAMessageStore

```
<virtualhost>
  <name>myhost</name>
  <myvhost>
    <store>
      <class>org.apache.qpid.server.store.berkeleydb.BDBHAMessageStore</class>
      <environment-path>${work}/bdbhastore</environment-path>
      <highAvailability>
        <groupName>myclustername</groupName>
        <nodeName>mynode1</nodeName>
        <nodeHostPort>node1host:port</nodeHostPort>
        <helperHostPort>node1host:port</helperHostPort>
        <durability>NO_SYNC\,NO_SYNC\,SIMPLE_MAJORITY</durability>
        <coalescingSync>true|false</coalescingSync>
        <designatedPrimary>true|false</designatedPrimary>
      </highAvailability>
    </store>
    ...
 </myvhost>
</virtualhost>
```

The `groupName` is the name of logical name of the cluster. All nodes within the cluster must use the same `groupName` in order to be considered part of the cluster.

The `nodeName` is the logical name of the node. All nodes within the cluster must have a unique name. It is recommended that the node name should be chosen from a different nomenclature from that of the servers on which they are hosted, in case the need arises to move node to a new server in the future.

The `nodeHostPort` is the hostname and port number used by this node to communicate with the the other nodes in the cluster. For the hostname, an IP address, hostname or fully qualified hostname may be used. For the port number, any free port can be used. It is important that this address is stable over time, as BDB records and uses this address internally.

The `helperHostPort` is the hostname and port number that new nodes use to discover other nodes within the cluster when they are newly introduced to the cluster. When configuring the first node, set the `helperHostPort` to its own `nodeHostPort`. For the second and subsequent nodes, set their `helperHostPort` to that of the first node.

`durability` controls the durability guarantees made by the cluster. It is important that all nodes use the same value for this property. The default value is NO_SYNC\,NO_SYNC\,SIMPLE_MAJORITY. Owing to the internal use of Apache Commons Config, it is currently necessary to escape the commas within the durability string.

`coalescingSync` controls the coalescing-sync mode of Qpid. It is important that all nodes use the same value. If omitted, it defaults to true.

The `designatedPrimary` is applicable only to the two-node case. It governs the behaviour of a node when the other node fails or becomes uncontactable. If true, the node will be designated as primary at startup and will be able to continue operating as a single node master. If false, the node will transition to an unavailable state until a third-party manually designates the node as primary or the other node is restored. It is suggested that the node that normally fulfils the role of master is set true in config file and the node that is normally replica is set false. Be aware that setting both nodes to true will lead to a failure to start up, as both cannot be designated at the point of contact. Designating both nodes as primary at runtime (using the JMX interface) will lead to a split-brain in the case of network partition and must be avoided.

Note

Usage of domain names in `helperHostPort` and `nodeHostPort` is more preferebale over IP addresses due to the tendency of more frequent changes of the last over the former. If server IP address changes but domain name remains the same the HA cluster can continue working as normal in case when domain names are used in cluster configuration. In case when IP addresses are used and they are changed with the time than Qpid JMX API for HA can be used to change the addresses or remove the nodes from the cluster.

12.5.1. Passing BDB environment and replication configuration options

It is possible to pass BDB environment [http://docs.oracle.com/cd/E17277_02/html/java/com/sleepycat/je/EnvironmentConfig.html] and replication [http://docs.oracle.com/cd/E17277_02/html/java/com/sleepycat/je/rep/ReplicationConfig.html] configuration options from the virtualhost.xml. Environment configuration options are passed using the `envConfig` element, and replication config using `repConfig`.

For example, to override the BDB environment configuration options `je.cleaner.threads` and `je.txn.timeout`

```
      ...
    </highAvailability>
    <envConfig>
      <name>je.cleaner.threads</name>
      <value>2</value>
    </envConfig>
    <envConfig>
      <name>je.txn.timeout</name>
      <value>15 min</value>
    </envConfig>
    ...
  </store>
```

And to override the BDB replication configuration options `je.rep.electionsPrimaryRetries`.

```
      ...
    </highAvailability>
    ...
    <repConfig>
```

```
      <name>je.rep.electionsPrimaryRetries</name>
      <value>3</value>
    </repConfig>
    ...
</store>
```

12.6. Durability Guarantees

The term durability [http://en.wikipedia.org/wiki/ACID#Durability] is used to mean that once a transaction is committed, it remains committed regardless of subsequent failures. A highly durable system is one where loss of a committed transaction is extermely unlikely, whereas with a less durable system loss of a transaction is likely in a greater number of scenarios. Typically, the more highly durable a system the slower and more costly it will be.

Qpid exposes the all the durability controls [http://oracle.com/cd/E17277_02/html/ReplicationGuide/txn-management.html#durabilitycontrols] offered by by BDB JE JA and a Qpid specific optimisation called **coalescing-sync** which defaults to enabled.

12.6.1. BDB Durability Controls

BDB expresses durability as a triplet with the following form:

```
<master sync policy>,<replica sync policy>,<replica acknowledgement policy>
```

The sync polices controls whether the thread performing the committing thread awaits the successful completion of the write, or the write and sync before continuing. The master sync policy and replica sync policy need not be the same.

For master and replic sync policies, the available values are: SYNC [http://docs.oracle.com/cd/E17277_02/html/java/com/sleepycat/je/Durability.SyncPolicy.html#SYNC], WRITE_NO_SYNC [http://docs.oracle.com/cd/E17277_02/html/java/com/sleepycat/je/Durability.SyncPolicy.html#WRITE_NO_SYNC], NO_SYNC [http://docs.oracle.com/cd/E17277_02/html/java/com/sleepycat/je/Durability.SyncPolicy.html#NO_SYNC]. SYNC is offers the highest durability whereas NO_SYNC the lowest.

Note: the combination of a master sync policy of SYNC and coalescing-sync true would result in poor performance with no corresponding increase in durability guarantee. It cannot not be used.

The acknowledgement policy defines whether when a master commits a transaction, it also awaits for the replica(s) to commit the same transaction before continuing. For the two-node case, ALL and SIMPLE_MAJORITY are equal.

For acknowledgement policy, the available value are: ALL [http://docs.oracle.com/cd/E17277_02/html/java/com/sleepycat/je/Durability.ReplicaAckPolicy.html#ALL], SIMPLE_MAJORITY [http://docs.oracle.com/cd/E17277_02/html/java/com/sleepycat/je/Durability.ReplicaAckPolicy.html#SIMPLE_MAJORITY] NONE [http://docs.oracle.com/cd/E17277_02/html/java/com/sleepycat/je/Durability.ReplicaAckPolicy.html#NONE].

12.6.2. Coalescing-sync

If enabled (the default) Qpid works to reduce the number of separate file-system sync [http://oracle.com/javase/6/docs/api/java/io/FileDescriptor.html#sync()] operations performed by the **master** on the underlying storage device thus improving performance. It does this coalescing separate sync operations arising from the different client commits operations occuring at approximately the same time. It does this in such a manner not to reduce the ACID guarantees of the system.

Coalescing-sync has no effect on the behaviour of the replicas.

12.6.3. Default

The default durability guarantee is `NO_SYNC, NO_SYNC, SIMPLE_MAJORITY` with coalescing-sync enabled. The effect of this combination is described in the table below. It offers a good compromise between durability guarantee and performance with writes being guaranteed on the master and the additional guarantee that a majority of replicas have received the transaction.

12.6.4. Examples

Here are some examples illustrating the effects of the durability and coalescing-sync settings.

Table 12.1. Effect of different durability guarantees

	Durability	**Coalescing-sync**	**Description**
1	NO_SYNC, NO_SYNC, SIMPLE_MAJORITY	true	Before the commit returns to the client, the transaction will be written/sync'd to the Master's disk (effect of coalescing-sync) and a majority of the replica(s) will have acknowledged the **receipt** of the transaction. The replicas will write and sync the transaction to their disk at a point in the future governed by ReplicationMutableConfig#LOG_FLUSH_ [http://docs.oracle.com/cd/E17277_02/html/java/com/sleepycat/je/rep/ReplicationMutableConfig.html#LOG_FLU
2	NO_SYNC, WRITE_NO_SYNC, SIMPLE_MAJORITY	true	Before the commit returns to the client, the transaction will be written/sync'd to the Master's disk (effect of coalescing-sync and a majority of the replica(s) will have acknowledged the **write** of the transaction to their disk. The replicas will sync the transaction to disk at a point in the future with an upper bound

	Durability	Coalescing-sync	Description
			governed by ReplicationMutableConfig#LOG_FLUSH_
3	NO_SYNC, NO_SYNC, NONE	false	After the commit returns to the client, the transaction is neither guaranteed to be written to the disk of the master nor received by any of the replicas. The master and replicas will write and sync the transaction to their disk at a point in the future with an upper bound governed by ReplicationMutableConfig#LOG_FLUSH_ This offers the weakest durability guarantee.

12.7. Client failover configuration

The details about format of Qpid connection URLs can be found at section Connection URLs [../../Programming-In-Apache-Qpid/html/QpidJNDI.html] of book Programming In Apache Qpid [../../Programming-In-Apache-Qpid/html/].

The failover policy option in the connection URL for the HA Cluster should be set to *roundrobin*. The Master broker should be put into a first place in *brokerlist* URL option. The recommended value for *connectdelay* option in broker URL should be set to the value greater than 1000 milliseconds. If it is desired that clients re-connect automatically after a master to replica failure, `cyclecount` should be tuned so that the retry period is longer than the expected length of time to perform the failover.

Example 12.2. Example of connection URL for the HA Cluster

amqp://guest:guest@clientid/test?brokerlist='tcp://localhost:5672?connectdelay='2000'&retries='3';tcp://localhost:5671?connectdelay='2000'&retries='3';tcp://localhost:5673?connectdelay='2000'&retries='3''&failover='roundrobin?cyclecount='30''

12.8. Qpid JMX API for HA

Qpid exposes the BDB HA store information via its JMX interface and provides APIs to remove a Node from the group, update a Node IP address, and assign a Node as the designated primary.

An instance of the `BDBHAMessageStore` MBean is instantiated by the broker for the each virtualhost using the HA store.

The reference to this MBean can be obtained via JMX API using an ObjectName like *org.apache.qpid:type=BDBHAMessageStore,name="<virtualhost name>"* where <virtualhost name> is the name of a specific virtualhost on the broker.

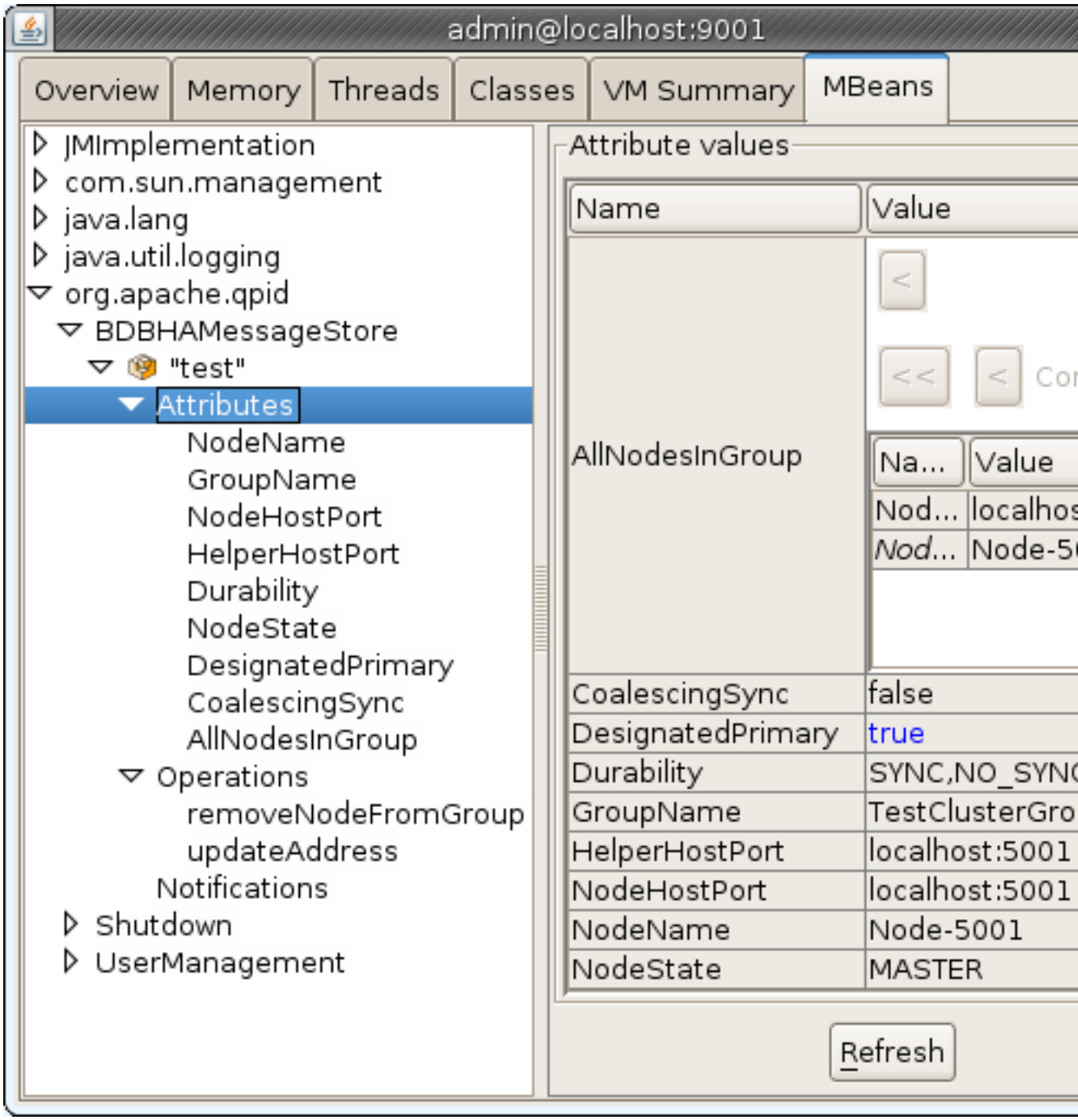
Table 12.2. Mbean `BDBHAMessageStore` attributes

Name	Type	Accessibility	Description
GroupName	String	Read only	Name identifying the group
NodeName	String	Read only	Unique name identifying the node within the group
NodeHostPort	String	Read only	Host/port used to replicate data between this node and others in the group
HelperHostPort	String	Read only	Host/port used to allow a new node to discover other group members
NodeState	String	Read only	Current state of the node
ReplicationPolicy	String	Read only	Node replication durability
DesignatedPrimary	boolean	Read/Write	Designated primary flag. Applicable to the two node case.
CoalescingSync	boolean	Read only	Coalescing sync flag. Applicable to the master sync policies NO_SYNC and WRITE_NO_SYNC only.
getAllNodesInGroup	TabularData	Read only	Get all nodes within the group, regardless of whether currently attached or not

Table 12.3. Mbean `BDBHAMessageStore` operations

Operation	Parameters	Returns	Description
removeNodeFromGroup	*nodeName*, name of node, string	void	Remove an existing node from the group
updateAddress	• *nodeName*, name of node, string • *newHostName*, new host name, string • *newPort*, new port number, int	void	Update the address of another node. The node must be in a STOPPED state.

Figure 12.7. BDBHAMessageStore view from jconsole.



Example 12.3. Example of java code to get the node state value

```
Map<String, Object> environment = new HashMap<String, Object>();

// credentials: user name and password
environment.put(JMXConnector.CREDENTIALS, new String[] {"admin","admin"});
JMXServiceURL url =  new JMXServiceURL("service:jmx:rmi:///jndi/rmi://localhost:90
JMXConnector jmxConnector = JMXConnectorFactory.connect(url, environment);
MBeanServerConnection mbsc =  jmxConnector.getMBeanServerConnection();

ObjectName queueObjectName = new ObjectName("org.apache.qpid:type=BDBHAMessageStor
String state = (String)mbsc.getAttribute(queueObjectName, "NodeState");

System.out.println("Node state:" + state);
```

Example system output:

```
Node state:MASTER
```

12.9. Monitoring cluster

In order to discover potential issues with HA Cluster early, all nodes in the Cluster should be monitored on regular basis using the following techniques:

- Broker log files scrapping for WARN or ERROR entries and operational log entries like:
  - *MST-1007 :* Store Passivated. It can indicate that Master virtual host has gone down.
  - *MST-1006 :* Recovery Complete. It can indicate that a former Replica virtual host is up and became the Master.
- Disk space usage and system load using system tools.
- Berkeley HA node status using `DbPing` [http://docs.oracle.com/cd/E17277_02/html/java/com/sleepycat/je/rep/util/DbPing.html] utility.

Example 12.4. Using `DbPing` utility for monitoring HA nodes.

java -jar je-5.0.73.jar DbPing -groupName TestClusterGroup -nodeName Node-5001 -nodeHost localhost:5001 -socketTimeout 10000

```
Current state of node: Node-5001 from group: TestClusterGroup
  Current state: MASTER
  Current master: Node-5001
  Current JE version: 5.0.73
  Current log version: 8
  Current transaction end (abort or commit) VLSN: 165
  Current master transaction end (abort or commit) VLSN: 0
  Current active feeders on node: 0
  Current system load average: 0.35
```

In the example above `DbPing` utility requested status of Cluster node with name *Node-5001* from replication group *TestClusterGroup* running on host *localhost:5001*. The state of the node was reported into a system output.

- Using Qpid broker JMX interfaces.

  Mbean `BDBHAMessageStore` can be used to request the following node information:

  - *NodeState* indicates whether node is a Master or Replica.
  - *Durability* replication durability.
  - *DesignatedPrimary* indicates whether Master node is designated primary.
  - *GroupName* replication group name.
  - *NodeName* node name.
  - *NodeHostPort* node host and port.
  - *HelperHostPort* helper host and port.
  - *AllNodesInGroup* lists of all nodes in the replication group including their names, hosts and ports.

  For more details about `BDBHAMessageStore` MBean please refer section Qpid JMX API for HA

12.10. Disk space requirements

Disk space is a critical resource for the HA Qpid broker.

In case when a Replica goes down (or falls behind the Master in 2 node cluster where the Master is designated primary) and the Master continues running, the non-replicated store files are kept on the Masters disk for the period of time as specified in *je.rep.repStreamTimeout* JE setting in order to replicate this data later when the Replica is back. This setting is set to 1 hour by default by the broker. The setting can be overridden as described in Section 12.5.1, “Passing BDB environment and replication configuration options”.

Depending from the application publishing/consuming rates and message sizes, the disk space might become overfull during this period of time due to preserved logs. Please, make sure to allocate enough space on your disk to avoid this from happening.

12.11. Network Requirements

The HA Cluster performance depends on the network bandwidth, its use by existing traffic, and quality of service.

In order to achieve the best performance it is recommended to use a separate network infrastructure for the Qpid HA Nodes which might include installation of dedicated network hardware on Broker hosts, assigning a higher priority to replication ports, installing a cluster in a separate network not impacted by any other traffic.

12.12. Security

At the moment Berkeley replication API supports only TCP/IP protocol to transfer replication data between Master and Replicas.

As result, the replicated data is unprotected and can be intercepted by anyone having access to the replication network.

Also, anyone who can access to this network can introduce a new node and therefore receive a copy of the data.

In order to reduce the security risks the entire HA cluster is recommended to run in a separate network protected from general access.

12.13. Backups

In order to protect the entire cluster from some cataclysms which might destroy all cluster nodes, backups of the Master store should be taken on a regular basis.

Qpid Broker distribution includes the "hot" backup utility *backup.sh* which can be found at broker bin folder. This utility can perform the backup when broker is running.

backup.sh script invokes `org.apache.qpid.server.store.berkeleydb.BDBBackup` to do the job.

You can also run this class from command line like in an example below:

Example 12.5. Performing store backup by using `BDBBackup` class directly

java -cp qpid-bdbstore-0.18.jar org.apache.qpid.server.store.berkeleydb.BDBBackup -fromdir path/to/store/folder -todir path/to/backup/foldeAr

In the example above BDBBackup utility is called from qpid-bdbstore-0.18.jar to backup the store at *path/to/store/folder* and copy store logs into *path/to/backup/folder*.

Linux and Unix users can take advantage of *backup.sh* bash script by running this script in a similar way.

Example 12.6. Performing store backup by using `backup.sh` bash script

backup.sh -fromdir path/to/store/folder -todir path/to/backup/folder

Note

Do not forget to ensure that the Master store is being backed up, in the event the Node elected Master changes during the lifecycle of the cluster.

12.14. Migration of a non-HA store to HA

Non HA stores starting from schema version 4 (0.14 Qpid release) can be automatically converted into HA store on broker startup if replication is first enabled with the

`DbEnableReplication` [http://docs.oracle.com/cd/E17277_02/html/java/com/sleepycat/je/rep/util/DbEnableReplication.html] utility from the BDB JE jar.

DbEnableReplication converts a non HA store into an HA store and can be used as follows:

Example 12.7. Enabling replication

java -jar je-5.0.73.jar DbEnableReplication -h /path/to/store -groupName MyReplicationGroup -nodeName MyNode1 -nodeHostPort localhost:5001

In the examples above, je jar of version 5.0.73 is used to convert store at */path/to/store* into HA store having replication group name *MyReplicationGroup*, node name *MyNode1* and running on host *localhost* and port *5001*.

After running DbEnableReplication and updating the virtual host store to configuration to be an HA message store, like in example below, on broker start up the store schema will be upgraded to the most recent version and the broker can be used as normal.

Example 12.8. Example of XML configuration for HA message store

```
<store>
    <class>org.apache.qpid.server.store.berkeleydb.BDBHAMessageStore</class>
    <environment-path>/path/to/store</environment-path>
    <highAvailability>
        <groupName>MyReplicationGroup</groupName>
        <nodeName>MyNode1</nodeName>
        <nodeHostPort>localhost:5001</nodeHostPort>
        <helperHostPort>localhost:5001</helperHostPort>
    </highAvailability>
</store>
```

The Replica nodes can be started with empty stores. The data will be automatically copied from Master to Replica on Replica start-up. This will take a period of time determined by the size of the Masters store and the network bandwidth between the nodes.

Note

Due to existing caveats in Berkeley JE with copying of data from Master into Replica it is recommended to restart the Master node after store schema upgrade is finished before starting the Replica nodes.

12.15. Disaster Recovery

This section describes the steps required to restore HA broker cluster from backup.

The detailed instructions how to perform backup on replicated environment can be found here.

At this point we assume that backups are collected on regular basis from Master node.

Replication configuration of a cluster is stored internally in HA message store. This information includes IP addresses of the nodes. In case when HA message store needs to be restored on a different host with a different IP address the cluster replication configuration should be reseted in this case

Oracle provides a command line utility `DbResetRepGroup` [http://docs.oracle.com/cd/E17277_02/html/java/com/sleepycat/je/rep/util/DbResetRepGroup.html] to reset the members of a replication group and replace the group with a new group consisting of a single new member as described by the arguments supplied to the utility

Cluster can be restored with the following steps:

- Copy log files into the store folder from backup
- Use `DbResetRepGroup` to reset an existing environment. See an example below

Example 12.9. Reseting of replication group with `DbResetRepGroup`

java -cp je-5.0.73.jar com.sleepycat.je.rep.util.DbResetRepGroup -h ha-work/Node-5001/bdbstore -groupName TestClusterGroup -nodeName Node-5001 -nodeHostPort localhost:5001

In the example above `DbResetRepGroup` utility from Berkeley JE of version 5.0.73 is used to reset the store at location *ha-work/Node-5001/bdbstore* and set a replication group to *TestClusterGroup* having a node *Node-5001* which runs at *localhost:5001*.

- Start a broker with HA store configured as specified on running of `DbResetRepGroup` utility.
- Start replica nodes having the same replication group and a helper host port pointing to a new master. The store content will be copied into Replicas from Master on their start up.

12.16. Performance

The aim of this section is not to provide exact performance metrics relating to HA, as this depends heavily on the test environment, but rather showing an impact of HA on Qpid broker performance in comparison with the Non HA case.

For testing of impact of HA on a broker performance a special test script was written using Qpid performance test framework. The script opened a number of connections to the Qpid broker, created producers and consumers on separate connections, and published test messages with concurrent producers into a test queue and consumed them with concurrent consumers. The table below shows the number of producers/consumers used in the tests. The overall throughput was collected for each configuration.

Table 12.4. Number of producers/consumers in performance tests

Test	Number of producers	Number of consumers
1	1	1
2	2	2
3	4	4
4	8	8
5	16	16
6	32	32
7	64	64

The test was run against the following Qpid Broker configurations

- Non HA Broker

- HA 2 Nodes Cluster with durability *SYNC,SYNC,ALL*
- HA 2 Nodes Cluster with durability *WRITE_NO_SYNC,WRITE_NO_SYNC,ALL*
- HA 2 Nodes Cluster with durability *WRITE_NO_SYNC,WRITE_NO_SYNC,ALL* and *coalescing-sync* Qpid mode
- HA 2 Nodes Cluster with durability *WRITE_NO_SYNC,NO_SYNC,ALL* and *coalescing-sync* Qpid mode
- HA 2 Nodes Cluster with durability *NO_SYNC,NO_SYNC,ALL* and *coalescing-sync* Qpid option

The evironment used in testing consisted of 2 servers with 4 CPU cores (2x Intel(r) Xeon(R) CPU 5150@2.66GHz), 4GB of RAM and running under OS Red Hat Enterprise Linux AS release 4 (Nahant Update 4). Network bandwidth was 1Gbit.

We ran Master node on the first server and Replica and clients(both consumers and producers) on the second server.

In non-HA case Qpid Broker was run on a first server and clients were run on a second server.

The table below contains the test results we measured on this environment for different Broker configurations.

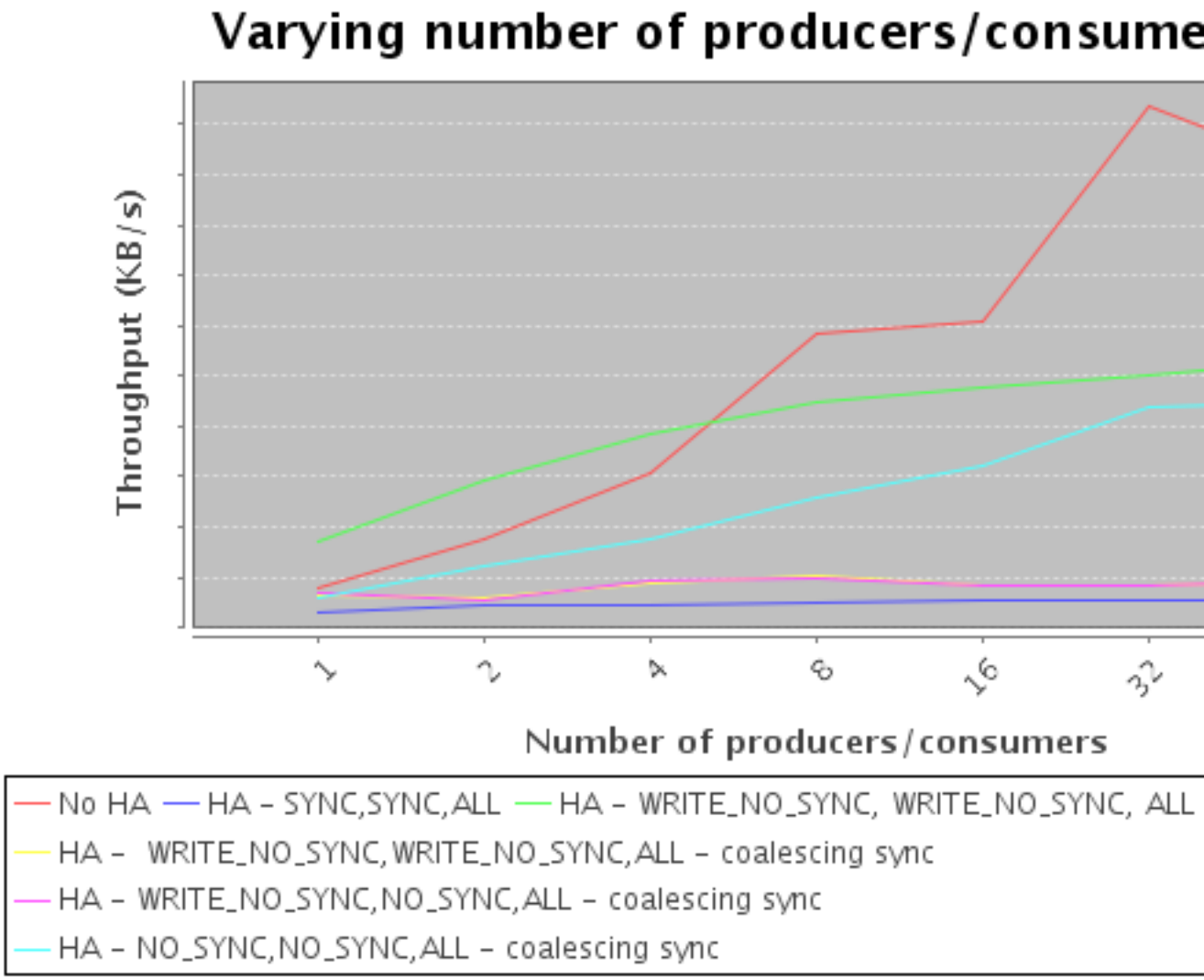
Each result is represented by throughput value in KB/second and difference in % between HA configuration and non HA case for the same number of clients.

Table 12.5. Performance Comparison

Test/Broker	No HA	SYNC, SYNC, ALL	WRITE_NO_SYNC, WRITE_NO_SYNC, ALL	WRITE_NO_SYNC, WRITE_NO_SYNC, ALL - coalescing-sync	WRITE_NO_SYNC, SYNC, ALL - coalescing-sync	NO_SYNC, NO_SYNC, ALL - coalescing-sync
1 (1/1)	0.0%	-61.4%	117.0%	-16.02%	-9.58%	-25.47%
2 (2/2)	0.0%	-75.43%	67.87%	-66.6%	-69.02%	-30.43%
3 (4/4)	0.0%	-84.89%	24.19%	-71.02%	-69.37%	-43.67%
4 (8/8)	0.0%	-91.17%	-22.97%	-82.32%	-83.42%	-55.5%
5 (16/16)	0.0%	-91.16%	-21.42%	-86.6%	-86.37%	-46.99%
6 (32/32)	0.0%	-94.83%	-51.51%	-92.15%	-92.02%	-57.59%
7 (64/64)	0.0%	-94.2%	-41.84%	-89.55%	-89.55%	-50.54%

The figure below depicts the graphs for the performance test results

Figure 12.8. Test results



On using durability *SYNC,SYNC,ALL* (without coalescing-sync) the performance drops significantly (by 62-95%) in comparison with non HA broker.

Whilst, on using durability *WRITE_NO_SYNC,WRITE_NO_SYNC,ALL* (without coalescing-sync) the performance drops by only half, but with loss of durability guarantee, so is not recommended.

In order to have better performance with HA, Qpid Broker comes up with the special mode called coalescing-sync, With this mode enabled, Qpid broker batches the concurrent transaction commits and syncs transaction data into Master disk in one go. As result, the HA performance only drops by 25-60% for durability *NO_SYNC,NO_SYNC,ALL* and by 10-90% for *WRITE_NO_SYNC,WRITE_NO_SYNC,ALL*.

Chapter 13. Miscellaneous

13.1. JVM Installation verification

13.1.1. Verify JVM on Windows

Firstly confirm that the JAVA_HOME environment variable is set correctly by typing the following at the command prompt:

```
echo %JAVA_HOME%
```

If JAVA_HOME is set you will see something similar to the following:

```
c:"\PROGRA~1"\Java\jdk1.6.0_24\
```

Then confirm that a Java installation (1.6 or higher) is available:

```
java -version
```

If java is available on the path, output similar to the following will be seen:

```
java version "1.6.0_24"
Java(TM) SE Runtime Environment (build 1.6.0_24-b07)
Java HotSpot(TM) 64-Bit Server VM (build 19.1-b02, mixed mode)
```

13.1.2. Verify JVM on Unix

Firstly confirm that the JAVA_HOME environment variable is set correctly by typing the following at the command prompt:

```
echo $JAVA_HOME
```

If JAVA_HOME is set you will see something similar to the following:

```
/usr/java/jdk1.6.0_35
```

Then confirm that a Java installation (1.6 or higher) is available:

```
java -version
```

If java is available on the path, output similar to the following will be seen:

```
java version "1.6.0_35"
Java(TM) SE Runtime Environment (build 1.6.0_35-b10-428-11M3811)
Java HotSpot(TM) 64-Bit Server VM (build 20.10-b01-428, mixed mode)
```