Using Broker Federation

Introduction

Please note: Whereas broker federation was introduced in the M3 milestone release, the discussion in this document is based on the richer capabilities of federation in the M4 release.

This document presents broker federation for the administrative user. For design and developer information, please see Federation Design Note.

What Is Broker Federation?

The Qpid C++ messaging broker supports broker federation, a mechanism by which large messaging networks can be built using multiple brokers. Some scenarios in which federation is useful:

- Connecting disparate locations across a wide area network. In this case full connectivity across the enterprise can be achieved while keeping local message traffic isolated to a single location.
- Departmental brokers that have a policy which controls the flow of inter-departmental message traffic.
- Scaling of capacity for expensive broker operations. High-function exchanges like the XML exchange can be replicated to scale performance.
- Co-Resident brokers. Some applications benefit from having a broker co-resident with the client. This is particularly true if the client produces data that must be delivered reliably but connectivity to the consumer(s) is non-reliable. In this case, a co-resident broker provides queueing and durability not available in the client alone.
- Bridging disjoint IP networks. Message brokers can be configured to allow message connectivity between networks where there is no IP connectivity. For example, an isolated, private IP network can have messaging connectivity to brokers in other outside IP networks.

The qpid-route Utility

The qpid-route command line utility is provided with the Qpid broker. This utility is used to configure federated networks of brokers and to view the status and topology of networks.

qpid-route accesses the managed brokers remotely. It does not need to be invoked from the same host on which the broker is running. If network connectivity permits, an entire enterprise can be configured from a single location.

In the following sections, federation concepts will be introduced and illustrated using qpid-route.

Links and Routes

Federation occurs when a link is established between two brokers and one or more routes are created within that link. A link is a transport level connection (tcp, rdma, ssl, etc.) initiated by one broker and accepted by another. The initiating broker assumes the role of client with regard to the connection. The accepting broker annotates the connection as being for federation but otherwise treats it as a normal client connection.

A route is associated with an AMQP session established over the link connection. There may be multiple routes sharing the same link. A route controls the flow of messages across the link between brokers. Routes always consist of a session and a subscription for consuming messages. Depending on the configuration, a route may have a private queue on the source broker with a binding to an exchange on that broker.

Routes are unidirectional. A single route provides for the flow of messages in one direction across a link. If bidirectional connectivity is required (and it almost always is), then a pair of routes must be created, one for each direction of message flow.

The qpid-route utility allows the administrator to configure and manage links and routes separately. However, when a route is created and a link does not already exist, qpid-route will automatically create the link. It is typically not necessary to create a link by itself. It is, however, useful to get a list of links and their connection status from a broker:

```
$ qpid-route link list localhost:10001
```

<table>
<thead>
<tr>
<th>Host</th>
<th>Port</th>
<th>Transport</th>
<th>Durable</th>
<th>State</th>
<th>Last Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>localhost</td>
<td>10002</td>
<td>tcp</td>
<td>N</td>
<td>Operational</td>
<td></td>
</tr>
<tr>
<td>localhost</td>
<td>10003</td>
<td>tcp</td>
<td>N</td>
<td>Operational</td>
<td></td>
</tr>
<tr>
<td>localhost</td>
<td>10009</td>
<td>tcp</td>
<td>N</td>
<td>Waiting</td>
<td>Connection refused</td>
</tr>
</tbody>
</table>

The example above shows a link list query to the broker at "localhost:10001". In the example, this broker has three links to other brokers. Two are operational and the third is waiting to connect because there is not currently a broker listening at that address.

The Life Cycle of a Link

When a link is created on a broker, that broker attempts to establish a transport-level connection to the peer broker. If it fails to connect, it retries the connection at an increasing time interval. If the connection fails due to authentication failure, it will not continue to retry as administrative intervention is needed to fix the problem.

If an operational link is disconnected, the initiating broker will attempt to re-establish the connection with the same interval back-off.
The shortest retry-interval is 2 seconds and the longest is 64 seconds. Once enough consecutive retries have occurred that the interval has grown to 64 seconds, the interval will then stay at 64 seconds.

**Durable Links and Routes**

If, when a link or a route is created using qpid-route, the `--durable` option is used, it shall be durable. This means that its life cycle shall span restarts of the broker. If the broker is shut down, when it is restarted, the link will be restored and will begin establishing connectivity.

A non-durable route can be created for a durable link but a durable route cannot be created for a non-durable link.

```
$ qpid-route dynamic add localhost:10003 localhost:10004 fed.topic
$ qpid-route dynamic add localhost:10003 localhost:10004 fed.topic2 --durable
Failed: Can't create a durable route on a non-durable link
```

In the above example, a transient (non-durable) dynamic route was created between localhost:10003 and localhost:10004. Because there was no link in place, a new transient link was created. The second command is attempting to create a durable route over the same link and is rejected as illegal.

**Dynamic Routing**

Dynamic routing provides the simplest configuration for a network of brokers. When configuring dynamic routing, the administrator need only express the logical topology of the network (i.e. which pairs of brokers are connected by a unidirectional route). Queue configuration and bindings are handled automatically by the brokers in the network.

Dynamic routing uses the **Distributed Exchange** concept. From the client's point of view, all of the brokers in the network collectively offer a single logical exchange that behaves the same as a single exchange in a single broker. Each client connects to its local broker and can bind its queues to the distributed exchange and publish messages to the exchange.

When a consuming client binds a queue to the distributed exchange, information about that binding is propagated to the other brokers in the network to ensure that any messages matching the binding will be forwarded to the client's local broker. Messages published to the distributed exchange are forwarded to other brokers only if there are remote consumers to receive the messages. The dynamic binding protocol ensures that messages are routed only to brokers with eligible consumers. This includes topologies where messages must make multiple hops to reach the consumer.

When creating a dynamic routing network, the type and name of the exchange must be the same on each broker. It is strongly recommended that dynamic routes **NOT** be created using the standard exchanges (that is unless all messaging is intended to be federated).

A simple, two-broker network can be configured by creating an exchange on each broker then a pair of dynamic routes (one for each direction of message flow):

Create exchanges:

```
$ qpid-config -a localhost:10003 add exchange topic fed.topic
$ qpid-config -a localhost:10004 add exchange topic fed.topic
```

Create dynamic routes:

```
$ qpid-route dynamic add localhost:10003 localhost:10004 fed.topic
$ qpid-route dynamic add localhost:10004 localhost:10003 fed.topic
```

Information about existing routes can be gotten by querying each broker individually:

```
$ qpid-route route list localhost:10003
localhost:10003 localhost:10004 fed.topic <dynamic>
$ qpid-route route list localhost:10004
localhost:10004 localhost:10003 fed.topic <dynamic>
```

A nicer way to view the topology is to use `qpid-route route map`. The argument to this command is a single broker that serves as an entry point. `qpid-route` will attempt to recursively find all of the brokers involved in federation relationships with the starting broker and map all of the routes it finds.
More extensive and realistic examples are supplied later in this document.

Static Routing

Dynamic routing provides simple, efficient, and automatic handling of the bindings that control routing as long as the configuration keeps within a set of constraints (i.e., exchanges of the same type and name, bidirectional traffic flow, etc.). However, there are scenarios where it is useful for the administrator to have a bit more control over the details. In these cases, static routing is appropriate.

Exchange Routes

An exchange route is like a dynamic route except that the exchange binding is statically set at creation time instead of dynamically tracking changes in the network.

When an exchange route is created, a private queue (auto-delete, exclusive) is declared on the source broker. The queue is bound to the indicated exchange with the indicated key and the destination broker subscribes to the queue with a destination of the indicated exchange. Since only one exchange name is supplied, this means that exchange routes require that the source and destination exchanges have the same name.

Static exchange routes are added and deleted using `qpid-route route add` and `qpid-route route del` respectively. The following example creates a static exchange route with a binding key of "global.#" on the default topic exchange:

```bash
$ qpid-route route add localhost:10001 localhost:10002 amq.topic global.#
```

The route can be viewed by querying the originating broker (the destination in this case, see discussion of push and pull routes for more on this):

```bash
$ qpid-route route list localhost:10001
localhost:10001 localhost:10002 amq.topic global.#
```

Alternatively, the route map feature can be used to view the topology:

```bash
$ qpid-route route map localhost:10001
```

Finding Linked Brokers:
- localhost:10001... Ok
- localhost:10002... Ok

Dynamic Routes:
- none found

Static Routes:
- localhost:10001(ex=amq.topic) <= localhost:10002(ex=amq.topic) key=global.#

This example causes messages delivered to the `amq.topic` exchange on broker `localhost:10002` that have a key that matches `global.#` (i.e., starts with the string "global.") to be delivered to the `amq.topic` exchange on broker `localhost:10001`. This delivery will occur regardless of whether there are any consumers on `localhost:10001` that will receive the messages.

Note that this is a uni-directional route. No messages will be forwarded in the opposite direction unless another static route is created in the other direction.

The following diagram illustrates the result, in terms of AMQP objects, of the example static exchange route. In this diagram, the exchanges, both named "amq.topic" exist prior to the creation of the route. The creation of the route causes the private queue, the binding, and the subscription of the queue to the destination to be created.
Queue Routes

A queue route causes the destination broker to create a subscription to a pre-existing, possibly shared, queue on the source broker. There's no requirement that the queue be bound to any particular exchange. Queue routes can be used to connect exchanges of different names and/or types. They can also be used to distribute or balance traffic across multiple destination brokers.

Queue routes are created and deleted using the `qpid-route queue add` and `qpid-route queue del` commands respectively. The following example creates a static queue route to a public queue called "public" that feeds the amq.fanout exchange on the destination:

Create a queue on the source broker:

```
$ qpid-config -a localhost:10002 add queue public
```

Create a queue route to the new queue

```
$ qpid-route queue add localhost:10001 localhost:10002 amq.fanout public
```

Pull vs. Push Routes

When qpid-route creates or deletes a route, it establishes a connection to one of the brokers involved in the route and configures that broker. The configured broker then takes it upon itself to contact the other broker and exchange whatever information is needed to complete the setup of the route.

The notion of pull vs. pull is concerned with whether the configured broker is the source or the destination. The normal case is the pull route, where qpid-route configures the destination to pull messages from the source. A push route occurs when qpid-route configures the source to push messages to the destination.

Dynamic routes are always pull routes. Static routes are normally pull routes but may be inverted by using the `src-local` option when creating (or deleting) a route. If `src-local` is specified, qpid-route will make its connection to the source broker rather than the destination and configure the route to push rather than pull.

Push routes are useful in applications where brokers are co-resident with data sources and are configured to send data to a central broker. Rather than configure the central broker for each source, the sources can be configured to send to the destination.

qpid-route Summary and Options
$ qpid-route

Usage:  qpid-route [OPTIONS] dynamic add <dest-broker> <src-broker> <exchange> [tag] [exclude-list]
        qpid-route [OPTIONS] dynamic del <dest-broker> <src-broker> <exchange>
        qpid-route [OPTIONS] route add  <dest-broker> <src-broker> <exchange> <routing-key> [tag] [exclude-list]
        qpid-route [OPTIONS] route del  <dest-broker> <src-broker> <exchange> <routing-key>
        qpid-route [OPTIONS] route flush [dest-broker]
        qpid-route [OPTIONS] route map   [<broker>]
        qpid-route [OPTIONS] link add   <dest-broker> <src-broker>
        qpid-route [OPTIONS] link del  <dest-broker> <src-broker>
        qpid-route [OPTIONS] queue add   <dest-broker> <src-broker> <queue>
        qpid-route [OPTIONS] queue del  <dest-broker> <src-broker> <queue>

Options:
  --timeout seconds (10)   Maximum time to wait for broker connection
  -v [ --verbose ]         Verbose output
  -q [ --quiet ]           Quiet output, don't print duplicate warnings
  -d [ --durable ]         Added configuration shall be durable
  -e [ --del-empty-link ]  Delete link after deleting last route on the link
  -s [ --src-local ]       Make connection to source broker (push route)
  --ack N                  Acknowledge transfers over the bridge in batches of N
  -t <transport> [ --transport <transport>]  Specify transport to use for links, defaults to tcp

dest-broker and src-broker are in the form:  [username/password@] hostname | ip-address [:<port>]
ex: localhost, 10.1.1.7:10000, broker-host:10000, guest/guest@localhost

There are several transport options available for the federation link:

<table>
<thead>
<tr>
<th>Transport</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tcp</td>
<td>(default) A cleartext TCP connection</td>
</tr>
<tr>
<td>ssl</td>
<td>A secure TLS/SSL over TCP connection</td>
</tr>
<tr>
<td>rdma</td>
<td>A Connection using the RDMA interface (typically for an Infiniband network)</td>
</tr>
</tbody>
</table>

The tag and exclude-list arguments are not needed. They have been left in place for backward compatibility and for advanced users who might have very unusual requirements. If you're not sure if you need them, you don't. Leave them alone. If you must know, please refer to "Message Loop Prevention" in the advanced topics section below. The prevention of message looping is now automatic and requires no user action.

If the link between the two sites has network latency, this can be compensated for by increasing the ack frequency with --ack N to achieve better batching across the link between the two sites.

Caveats, Limitations, and Things to Avoid

Redundant Paths

The current implementation of federation in the M4 broker imposes constraints on redundancy in the topology. If there are parallel paths from a producer to a consumer, multiple copies of messages may be received.

A future release of Qpid will solve this problem by allowing redundant paths with cost metrics. This will allow the deployment of networks that are tolerant of connection or broker loss.

Lack of Flow Control

M4 broker federation uses unlimited flow control on the federation sessions. Flow control back-pressure will not be applied on inter-broker subscriptions.

Lack of Cluster Failover Support

The client functionality embedded in the broker for inter-broker links does not currently support cluster fail-over. This will be added in a subsequent release.

Example Scenarios
Using QPID to bridge disjoint IP networks

Multi-tiered topology

This topology can be configured using the following script.

```bash
## Define URLs for the brokers
broker1=localhost:10001
broker2=localhost:10002
broker3=localhost:10003
broker4=localhost:10004
broker5=localhost:10005
broker6=localhost:10006
broker7=localhost:10007
broker8=localhost:10008

## Create Topic Exchanges
qpid-config -a $broker1 add exchange topic fed.topic
qpid-config -a $broker2 add exchange topic fed.topic
qpid-config -a $broker3 add exchange topic fed.topic
qpid-config -a $broker4 add exchange topic fed.topic
qpid-config -a $broker5 add exchange topic fed.topic
qpid-config -a $broker6 add exchange topic fed.topic
qpid-config -a $broker7 add exchange topic fed.topic
qpid-config -a $broker8 add exchange topic fed.topic

## Create Topic Routes
qpid-route dynamic add $broker1 $broker2 fed.topic
qpid-route dynamic add $broker2 $broker1 fed.topic
qpid-route dynamic add $broker3 $broker2 fed.topic
qpid-route dynamic add $broker2 $broker3 fed.topic
qpid-route dynamic add $broker4 $broker2 fed.topic
qpid-route dynamic add $broker2 $broker4 fed.topic
qpid-route dynamic add $broker2 $broker5 fed.topic
qpid-route dynamic add $broker5 $broker2 fed.topic
qpid-route dynamic add $broker5 $broker6 fed.topic
qpid-route dynamic add $broker6 $broker5 fed.topic
qpid-route dynamic add $broker6 $broker7 fed.topic
qpid-route dynamic add $broker7 $broker6 fed.topic
qpid-route dynamic add $broker6 $broker8 fed.topic
qpid-route dynamic add $broker8 $broker6 fed.topic
```
Load-sharing across brokers

Advanced Topics

Federation Queue Naming
Message Loop Prevention