Red Hat AMQ 2020.Q4

Using AMQ Interconnect

For Use with AMQ Interconnect 1.9

Last Updated: 2020-10-08

Red Hat AMQ 2020.Q4 Using AMQ Interconnect

For Use with AMQ Interconnect 1.9

Legal Notice

Abstract

This guide describes how to install, configure, and manage AMQ Interconnect to build a large-scale messaging network.

Table of Contents

PART I. OVERVIEW

CHAPTER 1. OVERVIEW OF AMQ INTERCONNECT

AMQ Interconnect is a lightweight AMQP message router for building scalable, available, and performant messaging networks.

1.1. KEY FEATURES

You can use AMQ Interconnect to flexibly route messages between any AMQP-enabled endpoints, including clients, servers, and message brokers. AMQ Interconnect provides the following benefits:

- Connects clients and message brokers into an internet-scale messaging network with uniform addressing
- Supports high-performance direct messaging
- Uses redundant network paths to route around failures
- Streamlines the management of large deployments

1.2. SUPPORTED STANDARDS AND PROTOCOLS

AMQ Interconnect supports the following industry-recognized standards and network protocols:

- Version 1.0 of the Advanced Message Queueing Protocol (AMQP)
- Modern TCP with IPv6

NOTE

The details of distributed transactions (XA) within AMQP are not provided in the 1.0 version of the specification. AMQ Interconnect does not support XA transactions.

Additional resources

- OASIS AMQP 1.0 Specification .

1.3. SUPPORTED CONFIGURATIONS

AMQ Interconnect is supported on Red Hat Enterprise Linux 6, 7, and 8. See Red Hat AMQ 7 Supported Configurations for more information.

1.4. DOCUMENT CONVENTIONS

In this document, **sudo** is used for any command that requires root privileges. You should always exercise caution when using **sudo**, as any changes can affect the entire system.

For more information about using **sudo**, see The **sudo** Command.

PART II. LEARN

CHAPTER 2. IMPORTANT TERMS AND CONCEPTS

Before using AMQ Interconnect, you should be familiar with AMQP and understand some key concepts about AMQ Interconnect.

2.1. OVERVIEW OF AMQP

AMQ Interconnect implements version 1.0 of the Advanced Message Queueing Protocol (AMQP) specification. Therefore, you should understand several key AMQP terms and concepts before deploying or configuring AMQ Interconnect.

Containers

AMQP is a wire-level messaging protocol for transferring messages between applications called *containers*. In AMQP, a container is any application that sends or receives messages, such as a client application or message broker.
Containers connect to each other over *connections*, which are channels for communication.

Nodes

Containers contain addressable entities called *nodes* that are responsible for storing or delivering messages. For example, a queue on a message broker is a node.

Links

Messages are transferred between connected containers over *links*. A link is a unidirectional route between nodes. Essentially, a link is a channel for sending or receiving messages.
Links are established over *sesssions*, which are contexts for sending and receiving messages. Sessions are established over connections.

Additional resources

- OASIS AMQP 1.0 Specification
- AMQP Essentials Refcard
- Video series introducing AMQP 1.0

2.2. WHAT ROUTERS ARE

AMQ Interconnect is an application layer program running as a normal user program or as a daemon. A running instance of AMQ Interconnect is called a *router*.

Routers do not take responsibility for messages

Routers transfer messages between producers and consumers, but unlike message brokers, they do not take responsibility for messages. Instead, routers propagate message settlement and disposition across a network such that delivery guarantees are met. That is, the router network will deliver the message – possibly through several intermediate routers – and then route the consumer's acknowledgement of that message back across the same path. The responsibility for the message is transfered from the producer to the consumer as if they were directly connected.

Routers are combined to form router networks

Routers are often deployed in topologies of multiple routers called a router network. Routers use link-state routing protocols and algorithms similar to the Open Shortest Path First (OSPF) and Intermediate System to Intermediate System (IS-IS) protocols to calculate the best path from every message source to every message destination, and to recover quickly from failures. A router network relies on redundant network paths to provide continued connectivity in case of system or network failure.

Routers enhance both direct and indirect messaging patterns

A messaging client can make a single AMQP connection into a router network and, over that connection, exchange messages with one or more message brokers connected to any router in the network. At the same time, the client can exchange messages directly with other endpoints without involving a broker at all.

> **Example 2.1. Enhancing the use of message brokers**
>
> Routers can enhance a cluster of message brokers that provide a scalable, distributed work queue.
>
> The router network makes the broker cluster appear as a single queue, with producers publishing to a single address, and consumers subscribing to a single address. The router network can distribute work to any broker in the cluster, and collect work from any broker for any consumer.
>
> The routers improve the scalability of the broker cluster, because brokers can be added or removed from the cluster without affecting the clients.
>
> The routers also solve the common difficulty of "stuck messages". Without the router network, if a consumer is connected to a broker that does not have any messages (but other brokers in the cluster do have messages), you must either transfer the messages or leave them "stuck". The routers solve this issue, however, because all of the consumers are connected to all of the brokers through the router network. A message on any broker can be delivered to any of the consumers.

2.3. HOW ROUTERS ROUTE MESSAGES

In a router network, *routing* is the process by which messages are delivered to their destinations. To accomplish this, AMQ Interconnect offers two different routing mechanisms:

Message routing

Message routing enables you to distribute messages in anycast and multicast patterns. These patterns can be used for both direct routing, in which the router distributes messages between clients without a message broker, and indirect routing, in which the router enables clients to exchange messages through a message broker.
Message routing is useful for the following types of requirements:

- Default, basic message routing
  AMQ Interconnect automatically routes messages by default, so manual configuration is only required if you want routing behavior that is different than the default.

- Message-based routing patterns
  Message routing supports both anycast and multicast routing patterns. You can load-balance individual messages across multiple consumers, and multicast (or fan-out) messages to multiple subscribers.

- Sharding messages across multiple message brokers when message delivery order is not important
  Sharding messages from one producer might cause that producer’s messages to be received in a different order than the order in which they were sent.

Link routing

Link routing enables you to establish a dedicated, virtual "path" between a sender and receiver that travels through the router network. Link routes are typically used to connect clients to message brokers in scenarios in which a direct connection is unfeasible. Therefore, link routes enable messaging capabilities that are not possible with message routing, such as:

- Transactional messaging
  Link routing supports local transactions to a single broker. Distributed transactions are not supported.

- Guaranteed message delivery order
  Link routing to a sharded queue preserves the delivery order of the producer's messages by causing all messages on that link to go to the same broker instance.

- End-to-end flow control
  Flow control is "real" in that credits flow across the link route from the receiver to the sender.

- Server-side selectors
  With a link route, consumers can provide server-side selectors for broker subscriptions.

- Consumer specific acknowledgements
  With a link route, modified delivery states can be interpreted by a broker. For example, a broker can prevent redelivery of any messsages with the **undeliverable-here=true** modified delivery state.

Additional resources

- Section 12.1, "Configuring message routing"
- Section 12.2, "Creating link routes"

2.4. ROUTER SECURITY

AMQ Interconnect provides authentication and authorization mechanisms so that you can control who can access the router network, and what they can do with the messaging resources.

Authentication

AMQ Interconnect supports both SSL/TLS and SASL for encrypting and authenticating remote peers. Using these mechanisms, you can secure the router network in the following ways:

- Authenticate incoming connections from remote peers (such as clients and message brokers)
- Provide authentication credentials for outgoing connections to remote peers (such as clients and message brokers)
- Secure the inter-router connections between the routers in the router network

Authorization

AMQ Interconnect provides a **policy** mechanism that you can use to enforce user connection restrictions and AMQP resource access control.

Additional resources

- Chapter 9, *Securing network connections*
- Chapter 10, *Configuring authorization*

2.5. ROUTER MANAGEMENT

AMQ Interconnect provides both graphical and CLI tools for monitoring and managing a router network.

Red Hat AMQ Interconnect Console

A web console for monitoring the layout and health of the router network.

qdstat

A command-line tool for monitoring the status of a router in the router network. Using this tool, you can view the following information about a router:

- Incoming and outgoing connections
- Incoming and outgoing links
- Router network topology from the perspective of this router
- Addresses known to this router
- Link routes and autolinks
- Memory consumption information

qdmanage

A command-line tool for viewing and updating the configuration of a router at runtime.

Additional resources

- Management

PART III. GET STARTED

CHAPTER 3. GETTING STARTED

This section provides a quick introduction to AMQ Interconnect by showing you how to install AMQ Interconnect, start the router with the default configuration settings, and distribute messages between two clients.

3.1. INSTALLING AMQ INTERCONNECT ON RED HAT ENTERPRISE LINUX

AMQ Interconnect is distributed as a set of RPM packages, which are available through your Red Hat subscription.

Procedure

1. Ensure your subscription has been activated and your system is registered.
   For more information about using the Customer Portal to activate your Red Hat subscription and register your system for packages, see Appendix A, *Using your subscription*.

2. Subscribe to the required repositories:

   Red Hat Enterprise Linux 6

   ```
   $ sudo subscription-manager repos --enable=amq-interconnect-1-for-rhel-6-server-rpms --
   enable=amq-clients-2-for-rhel-6-server-rpms
   ```

   Red Hat Enterprise Linux 7

   ```
   $ sudo subscription-manager repos --enable=amq-interconnect-1-for-rhel-7-server-rpms --
   enable=amq-clients-2-for-rhel-7-server-rpms
   ```

   Red Hat Enterprise Linux 8

   ```
   $ sudo subscription-manager repos --enable=amq-interconnect-1-for-rhel-8-x86_64-rpms --
   enable=amq-clients-2-for-rhel-8-x86_64-rpms
   ```

3. Use the **yum** or **dnf** command to install the **qpid-dispatch-router**, **qpid-dispatch-tools**, and **qpid-dispatch-console** packages and their dependencies:

   ```
   $ sudo yum install qpid-dispatch-router qpid-dispatch-tools qpid-dispatch-console
   ```

4. Use the **which** command to verify that the **qdrouterd** executable is present.

   ```
   $ which qdrouterd
   /usr/sbin/qdrouterd
   ```

   The **qdrouterd** executable should be located at **/usr/sbin/qdrouterd**.

3.2. EXPLORING THE DEFAULT ROUTER CONFIGURATION FILE

The router’s configuration file (**qdrouterd.conf**) controls the way in which the router functions. The default configuration file contains the minimum number of settings required for the router to run. As you become more familiar with the router, you can add to or change these settings, or create your own configuration files.

By default, the router configuration file defines the following settings for the router:

- Operating mode
- How it listens for incoming connections
- Routing patterns for the message routing mechanism

Procedure

1. Open the following file: **/etc/qpid-dispatch/qdrouterd.conf**.
   When AMQ Interconnect is installed, **qdrouterd.conf** is installed in this directory. When the router is started, it runs with the settings defined in this file.

2. Review the default settings in **qdrouterd.conf**.

   Default configuration file

```
router {
    mode: standalone (1)
    id: Router.A (2)
}

listener { (3)
    host: 0.0.0.0
    port: amqp
    authenticatePeer: no
}

address { (4)
    prefix: closest
    distribution: closest
}

address {
    prefix: multicast
    distribution: multicast
}

address {
    prefix: unicast
    distribution: closest
}

address {
    prefix: exclusive
    distribution: closest
}

address {
    prefix: broadcast
    distribution: multicast
}
```

1. By default, the router operates in *standalone* mode. This means that it can only communicate with endpoints that are directly connected to it. It cannot connect to other routers, or participate in a router network.

2. The unique identifier of the router. This ID is used as the **container-id** (container name) at the AMQP protocol level. If it is not specified, the router shall generate a random identifier at startup.

3. The **listener** entity handles incoming connections from client endpoints. By default, the router listens on all network interfaces on the default AMQP port (5672).

4. By default, the router is configured to use the message routing mechanism. Each **address** entity defines how messages that are received with a particular address **prefix** should be distributed. For example, all messages with addresses that start with **closest** will be distributed using the **closest** distribution pattern.

NOTE

If a client requests a message with an address that is not defined in the router's configuration file, the **balanced** distribution pattern will be used automatically.

Additional resources

- For more information about the router configuration file (including available entities and attributes), see the qdrouterd man page.

3.3. STARTING THE ROUTER

After installing AMQ Interconnect, you start the router by using the **qdrouterd** command.

Procedure

1. Start the router:

```
$ qdrouterd
```

The router starts, using the default configuration file stored at **/etc/qpid-dispatch/qdrouterd.conf**.

2. Review the **qdrouterd** command output to verify the router status.
This example shows that the router was correctly installed, is running, and is ready to route traffic between clients:

```
$ qdrouterd
Fri May 20 09:38:03 2017 SERVER (info) Container Name: Router.A
Fri May 20 09:38:03 2017 ROUTER (info) Router started in Standalone mode
Fri May 20 09:38:03 2017 ROUTER (info) Router Core thread running. 0/Router.A
Fri May 20 09:38:03 2017 ROUTER (info) In-process subscription M/$management
Fri May 20 09:38:03 2017 AGENT (info) Activating management agent on
```

```
$_management_internal
Fri May 20 09:38:03 2017 ROUTER (info) In-process subscription L/$management
Fri May 20 09:38:03 2017 ROUTER (info) In-process subscription L/$_management_internal
Fri May 20 09:38:03 2017 DISPLAYNAME (info) Activating DisplayNameService on
$displayname
Fri May 20 09:38:03 2017 ROUTER (info) In-process subscription L/$displayname
Fri May 20 09:38:03 2017 CONN_MGR (info) Configured Listener: 0.0.0.0:amqp proto=any
role=normal
Fri May 20 09:38:03 2017 POLICY (info) Policy configured maximumConnections: 0,
policyFolder: '', access rules enabled: 'false'
Fri May 20 09:38:03 2017 POLICY (info) Policy fallback defaultApplication is disabled
Fri May 20 09:38:03 2017 SERVER (info) Operational, 4 Threads Running
```

Additional resources

- The qdrouterd man page.

3.4. SENDING TEST MESSAGES

After starting the router, send some test messages to see how the router can connect two endpoints by distributing messages between them.

This procedure demonstrates a simple configuration consisting of a single router with two clients connected to it: a sender and a receiver. The receiver wants to receive messages on a specific address, and the sender sends messages to that address.

A broker is not used in this procedure, so there is no *"store and forward"* mechanism in the middle. Instead, the messages flow from the sender, through the router, to the receiver only if the receiver is online, and the sender can confirm that the messages have arrived at their destination.

Prerequisites

AMQ Python must be installed. For more information, see Using the AMQ Python Client.

Procedure

1. Navigate to the AMQ Python examples directory.

   ```
   $ cd <install-dir>/examples/python/
   ```

   <install-dir>

   The directory where you installed AMQ Python.

2. Start the **simple_recv.py** receiver client.

   ```
   $ python simple_recv.py -a 127.0.0.1:5672/examples -m 5
   ```

   This command starts the receiver and listens on the **examples** address (**127.0.0.1:5672/examples**). The receiver is also set to receive a maximum of five messages.

   NOTE

   In practice, the order in which you start senders and receivers does not matter. In both cases, messages will be sent as soon as the receiver comes online.

3. In a new terminal window, navigate to the Python examples directory and run the **simple_send.py** example:

   ```
   $ cd <install-dir>/examples/python/
   $ python simple_send.py -a 127.0.0.1:5672/examples -m 5
   ```

   This command sends five auto-generated messages to the **examples** address (**127.0.0.1:5672/examples**) and then confirms that they were delivered and acknowledged by the receiver:

   ```
   all messages confirmed
   ```

4. Verify that the receiver client received the messages.
   The receiver client should display the contents of the five messages:

   ```
   {u'sequence': 1L}
   {u'sequence': 2L}
   {u'sequence': 3L}
   {u'sequence': 4L}
   {u'sequence': 5L}
   ```

3.5. NEXT STEPS

After using AMQ Interconnect to distribute messages between two clients, you can use the following sections to learn more about AMQ Interconnect configuration, deployment, and management.

Change the router’s configuration

AMQ Interconnect ships with default settings that are suitable for many basic use cases. You can further experiment with the standalone router that you used in the *Getting started* example by changing the router’s essential properties, network connections, security settings, logging, and routing mechanisms.

Install and configure AMQ Interconnect

AMQ Interconnect is typically deployed in router networks. You can design a router network of any arbitrary topology to interconnect the endpoints in your messaging network.

Monitor and manage AMQ Interconnect

You can use the web console and command-line management tools to monitor the status and performance of the routers in your router network.

PART IV. INSTALL

CHAPTER 4. AMQ INTERCONNECT DEPLOYMENT GUIDELINES

To plan your router network and design the network topology, you must first understand the different router modes and how you can use them to create different types of networks.

4.1. ROUTER OPERATING MODES

In AMQ Interconnect, each router can operate in *standalone*, *interior*, or *edge* mode. In a router network, you deploy multiple interior routers or a combination of interior and edge routers to create the desired network topology.

Standalone

The router operates as a single, standalone network node. A standalone router cannot be used in a router network - it does not establish connections with other routers, and only routes messages between directly-connected endpoints.

Interior

The router is part of the interior of the router network. Interior routers establish connections with each other and automatically compute the lowest cost paths across the network. You can have up to 128 interior routers in the router network.

Edge

The router maintains a single uplink connection to one or more interior routers. Edge routers do not participate in the routing protocol or route computation, but they enable you to efficiently scale the routing network. There are no limits to the number of edge routers you can deploy in a router network.

4.2. SECURITY GUIDELINES

In the router network, the interior routers should be secured with a strong authentication mechanism in which they identify themselves to each other. You should choose and plan this authentication mechanism before creating the router network.

WARNING

If the interior routers are not properly secured, unauthorized routers (or endpoints pretending to be routers) could join the router network, compromising its integrity and availability.

You can choose a security mechanism that best fits your requirements. However, you should consider the following recommendations:

- Create an X.509 Certificate Authority (CA) to oversee the interior portion of the router network.
- Generate an individual certificate for each interior router.
  Each interior router can be configured to use the CA to authenticate connections from any other interior routers.

NOTE

Connections from edge routers and clients can use different levels of security, depending on your requirements.

By using these recommendations, a new interior router cannot join the network until the owner of the CA issues a new certificate for the new router. In addition, an intruder wishing to spoof an interior router cannot do so because it would not have a valid X.509 certificate issued by the network's CA.

4.3. ROUTER CONNECTION GUIDELINES

Before creating a router network, you should understand how routers connect to each other, and the factors that affect the direction in which an inter-router connection should be established.

Inter-router connections are bidirectional

When a connection is established between routers, message traffic flows in both directions across that connection. Each connection has a client side (a *connector*) and a server side (a *listener*) for the purposes of connection establishment. Once the connection is established, the two sides become equal participants in a bidirectional connection. For the purposes of routing AMQP traffic across the network, the direction of connection establishment is not relevant.

Factors that affect the direction of connection establishment

When establishing inter-router connections, you must choose which router will be the "listener" and which will be the "connector". There should be only one connection between any pair of routers.

When determining the direction of inter-router connections in the network topology, consider the following factors:

IP network boundaries and firewalls

Generally, inter-router connections should always be established from more private to more public. For example, to connect a router in a private IP network to another router in a public location (such as a public cloud provider), the router in the private network must have the connector and the router in the public location must have the listener. This is because the public location cannot reach the private location by TCP/IP without the use of VPNs or other firewall features designed to allow public-to-private access.

Network topology

The topology of the router network may affect the direction in which connections should be established between the routers. For example, a star-topology that has a series of routers connected to one or two central "hub" routers should have listeners on the hub and connectors on the spokes. That way, new spoke routers may be added without changing the configuration of the hub.

CHAPTER 5. INSTALLING AMQ INTERCONNECT

You can deploy AMQ Interconnect as a single standalone router, or as multiple routers connected together in a router network. Router networks may represent any arbitrary topology, enabling you to design the network to best fit your requirements.

With AMQ Interconnect, the router network topology is independent from the message routing. This means that messaging clients always experience the same message routing behavior regardless of the underlying network topology. Even in a multi-site or hybrid cloud router network, the connected endpoints behave as if they were connected to a single, logical router.

To create the router network topology, complete the following:

1. Review the deployment guidelines.
   You should understand the different router operating modes you can deploy in your topology, and be aware of security requirements for the interior portion of the router network.

2. Install AMQ Interconnect on the host .
   If you are creating a router network with multiple routers, repeat this step on each host.

3. Prepare the router configurations.
   After installing AMQ Interconnect, configure it to define how it should connect to other routers and endpoints, and how it should operate.

4. Start the routers.
   After the routers are configured, start them so that they can connect to each other and begin routing messages.

5.1. INSTALLING AMQ INTERCONNECT ON RED HAT ENTERPRISE LINUX

AMQ Interconnect is distributed as a set of RPM packages, which are available through your Red Hat subscription.

Procedure

1. Ensure your subscription has been activated and your system is registered.
   For more information about using the Customer Portal to activate your Red Hat subscription and register your system for packages, see Appendix A, *Using your subscription*.

2. Subscribe to the required repositories:

   Red Hat Enterprise Linux 6

   ```
   $ sudo subscription-manager repos --enable=amq-interconnect-1-for-rhel-6-server-rpms --
   enable=amq-clients-2-for-rhel-6-server-rpms
   ```

   Red Hat Enterprise Linux 7

   ```
   $ sudo subscription-manager repos --enable=amq-interconnect-1-for-rhel-7-server-rpms --
   enable=amq-clients-2-for-rhel-7-server-rpms
   ```

   Red Hat Enterprise Linux 8

```
$ sudo subscription-manager repos --enable=amq-interconnect-1-for-rhel-8-x86_64-rpms --
enable=amq-clients-2-for-rhel-8-x86_64-rpms
```

3. Use the **yum** or **dnf** command to install the **qpid-dispatch-router**, **qpid-dispatch-tools**, and **qpid-dispatch-console** packages and their dependencies:

   ```
   $ sudo yum install qpid-dispatch-router qpid-dispatch-tools qpid-dispatch-console
   ```

4. Use the **which** command to verify that the **qdrouterd** executable is present.

   ```
   $ which qdrouterd
   /usr/sbin/qdrouterd
   ```

   The **qdrouterd** executable should be located at **/usr/sbin/qdrouterd**.

5.2. PREPARING ROUTER CONFIGURATIONS

After installing AMQ Interconnect, configure it to define how it should connect to other routers and endpoints, and how it should operate. If you are creating a router network, complete this workflow for each router in the network.

Prerequisites

- AMQ Interconnect is installed on the host.

Procedure

1. Configure essential router properties.
   To participate in a router network, a router must be configured with a unique ID and an operating mode.

2. Configure network connections.

   a. Connect the router to any other routers in the router network.
      Repeat this step for each additional router to which you want to connect this router.

   b. If the router should connect with an AMQP client, configure a client connection.

   c. If the router should connect to an external AMQP container (such as a message broker), configure the connection.

3. Secure each of the connections that you configured in the previous step .

4. (Optional) Configure any additional properties.
   These properties should be configured the same way on each router. Therefore, you should only configure each one once, and then copy the configuration to each additional router in the router network.

   - Authorization
     If necessary, configure policies to control which messaging resources clients are able to access on the router network.

   - Routing
     AMQ Interconnect automatically routes messages without any configuration: clients can

send messages to the router network, and the router automatically routes them to their destinations. However, you can configure the routing to meet your exact requirements. You can configure the routing patterns to be used for certain addresses, create waypoints and autolinks to route messages through broker queues, and create link routes to connect clients to brokers.

- Logging
  You can set the default logging configuration to ensure that events are logged at the correct level for your environment.

5. Repeat this workflow for each additional router that you want to add to the router network.

5.3. STARTING A ROUTER

You use the **qdrouterd** command to start a router. You can start a router in the foreground, the background, or as a service.

Procedure

- Do one of the following:

To…	Enter this command…
Start the router in the foreground	`$ qdrouterd`
Start the router in the background as a daemon	`$ qdrouterd -d`

Start the router as a service	**Red Hat Enterprise Linux 6**
	`$ sudo service qdrouterd start`
	Red Hat Enterprise Linux 7 and later versions
	`$ systemctl start qdrouterd.service`
	NOTE
	If you start the router as a service, the **systemd LimitNOFILE** limit affects the number of connections that can be open for the router. If you reach the limit, the router is not able to accept any more connections, and an error message is logged indicating "Too many open files". To avoid reaching this limit, increase the **LimitNOFILE** value for the **systemd** process.
	For more information, see How to set limits for services in RHEL 7 and systemd.

CHAPTER 6. UPGRADING AMQ INTERCONNECT

You should upgrade AMQ Interconnect to the latest version to ensure that you have the latest enhancements and fixes. The upgrade process involves installing the new AMQ Interconnect packages and restarting your routers.

You can use these instructions to upgrade AMQ Interconnect to a new *minor release* or *maintenance release*.

Minor Release

AMQ Interconnect periodically provides point releases, which are minor updates that include new features, as well as bug and security fixes. If you plan to upgrade from one AMQ Interconnect point release to another, for example, from AMQ Interconnect 1.0 to AMQ Interconnect 1.1, code changes should not be required for applications that do not use private, unsupported, or technical preview components.

Maintenance Release

AMQ Interconnect also periodically provides maintenance releases that contain bug fixes. Maintenance releases increment the minor release version by the last digit, for example from 1.0.0 to 1.0.1. A maintenance release should not require code changes; however, some maintenance releases might require configuration changes.

Prerequisites

Before performing an upgrade, you should have reviewed the release notes for the target release to ensure that you understand the new features, enhancements, fixes, and issues. To find the release notes for the target release, see the Red Hat Customer Portal .

Procedure

1. Upgrade the **qpid-dispatch-router** and **qpid-dispatch-tools** packages and their dependencies:

   ```
   $ sudo yum update qpid-dispatch-router qpid-dispatch-tools
   ```

   For more information, see Chapter 5, *Installing AMQ Interconnect* .

2. Restart each router in your router network.
   To avoid disruption, you should restart each router one at a time.

   This example restarts a router in Red Hat Enterprise Linux 7:

   ```
   $ systemctl restart qdrouterd.service
   ```

   For more information about starting a router, see Section 5.3, “Starting a router” .

PART V. CONFIGURE

CHAPTER 7. CONFIGURING ROUTER PROPERTIES

By default, AMQ Interconnect operates in **standalone** mode with a randomly-generated ID. If you want to use this router in a router network, you must change these properties.

Procedure

1. Open the **/etc/qpid-dispatch/qdrouterd.conf** configuration file.

2. In the **router** section, specify the mode and ID.
   This example shows a router configured to operate in **interior** mode:

   ```
   router {
       mode: interior
       id: Router.A
   }
   ```

   mode

   Specify one of the following modes:

   - **standalone** - Use this mode if the router does not communicate with other routers and is not part of a router network. When operating in this mode, the router only routes messages between directly connected endpoints.
   - **interior** - Use this mode if the router is part of a router network and needs to collaborate with other routers.
   - **edge** - Use this mode if the router is an edge router that will connect to a network of interior routers.

   id

   The unique identifier for the router. This ID will also be the container name at the AMQP protocol level.

3. If necessary, configure any additional properties for the router.
   For information about additional attributes, see router in the **qdrouterd.conf** man page.

CHAPTER 8. CONFIGURING NETWORK CONNECTIONS

AMQ Interconnect connects clients, servers, AMQP services, and other routers through network connections. To connect the router to other messaging endpoints, you configure *listeners* to accept connections, and *connectors* to make outbound connections. However, connections are bidirectional - once the connection is established, message traffic flows in both directions.

You can do the following:

- Connect a router to another router
- Listen for client connections
- Connect a router to an external AMQP container
- Add metadata to connections
- Understand connection failover

8.1. CONNECTING ROUTERS

To connect a router to another router in the router network, you configure a **connector** on one router to create the outbound connection, and a **listener** on the other router to accept the connection.

Because connections are bidirectional, there should only be one connection between any pair of routers. Once the connection is established, message traffic flows in both directions.

This procedure describes how to connect a router to another router in the router network.

Procedure

1. Determine the direction of the connection.
   Decide which router should be the "connector", and which should be the "listener". The direction of the connection establishment is sometimes arbitrary, but consider the following factors:

   IP network boundaries and firewalls

   Generally, inter-router connections should always be established from more private to more public. For example, to connect a router in a private IP network to another router in a public location (such as a public cloud provider), the router in the private network must be the "connector" and the router in the public location must be the "listener". This is because the public location cannot reach the private location by TCP/IP without the use of VPNs or other firewall features designed to allow public-to-private access.

   Network topology

   The topology of the router network may affect the direction in which connections should be established between the routers. For example, a star-topology that has a series of routers connected to one or two central "hub" routers should have "listeners" on the hub and "connectors" on the spokes. That way, new spoke routers may be added without changing the configuration of the hub.

2. On the router that should create the connection, open the **/etc/qpid-dispatch/qdrouterd.conf** configuration file and add a **connector**.
   This example creates a **connector** for an inter-router connection between two interior routers:

```
connector {
    host: 192.0.2.1
    port: 5001
    role: inter-router
    ...
}
```

host

The IP address (IPv4 or IPv6) or hostname on which the router will connect.

port

The port number or symbolic service name on which the router will connect.

role

The role of the connection. If the connection is between two interior routers, specify **inter-router**. If the connection is between an interior router and an edge router, specify **edge**.

3. On the router that should accept the connection establishment, open the **/etc/qpid-dispatch/qdrouterd.conf** configuration file and verify that an inter-router **listener** is configured.
   This example creates a **listener** to accept the connection establishment configured in the previous step:

```
listener {
    host: 0.0.0.0
    port: 5001
    role: inter-router
    ...
}
```

host

The IP address (IPv4 or IPv6) or hostname on which the router will listen.

port

The port number or symbolic service name on which the router will listen.

role

The role of the connection. If the connection is between two interior routers, specify **inter-router**. If the connection is between an interior router and an edge router, specify **edge**.

4. If the router should connect to any other routers, repeat this procedure.
   Edge routers can only connect to interior routers. They cannot connect to other edge routers.

Additional resources

- After connecting a router to another router, secure the connection.
  For more information, see Section 9.1, "Securing connections between routers" .

8.2. LISTENING FOR CLIENT CONNECTIONS

To enable a router to listen for and accept connections from AMQP clients, you configure a **listener**.

Once the connection is enabled on the router, clients can connect to it using the same methods they use to connect to a broker. From the client's perspective, the router connection and link establishment are identical to a broker connection and link establishment.

NOTE

Instead of configuring a **listener** to listen for connections from the client, you can configure a **connector** to initiate connections to the client. In this case, the router will use the **connector** to initiate the connection, but it will not create any links. Links are only created by the peer that accepts the connection.

Procedure

1. Open the **/etc/qpid-dispatch/qdrouterd.conf** configuration file.

2. Configure a **listener** with the **normal** role.

```
listener {
    host: primary.example.com
    port: 5672
    role: normal
    failoverUrls: secondary.example.com:20000, tertiary.example.com
    ...
}
```

host

The IP address (IPv4 or IPv6) or hostname on which the router will listen.

port

The port number or symbolic service name on which the router will listen.

role

The role of the connection. Specify **normal** to indicate that this connection is used for message delivery for AMQP clients.

failoverUrls (optional)

A comma-separated list of backup URLs the client can use to reconnect if the established connection is lost. Each URL must use the following form:
[(amqp|amqps|ws|wss)://](*HOST*|*IP ADDRESS*)[:port]

For more information, see Section 8.5, "Understanding connection failover" .

Additional resources

- After enabling a router to listen for client connections, secure the connection.
  For more information, see Section 9.2, "Securing incoming client connections" .

8.3. CONNECTING TO EXTERNAL AMQP CONTAINERS

To enable a router to establish a connection to an external AMQP container (such as a message broker), you configure a **connector**.

NOTE

Instead of configuring a **connector** to initiate connections to the AMQP container, you can configure a **listener** to listen for connections from the AMQP container. However, in this case, the addresses on the AMQP container are available for routing only after the AMQP container has created a connection.

Procedure

1. Open the **/etc/qpid-dispatch/qdrouterd.conf** configuration file.
2. Configure a **connector** with the **route-container** role.
   This example creates a **connector** that initiates connections to a broker. The addresses on the broker will be available for routing once the router creates the connection and it is accepted by the broker.

```
connector {
    name: my-broker
    host: 192.0.2.10
    port: 5672
    role: route-container
    ...
}
```

name

The name of the **connector**. Specify a name that describes the entity to which the router will connect.

host

The IP address (IPv4 or IPv6) or hostname to which the router will connect.

port

The port number or symbolic service name to which the router will connect.

role

The role of the connection. Specify **route-container** to indicate that this connection is for an AMQP container that holds known addresses.

Additional resources

- After enabling a router to connect to an external AMQP container, configure any necessary security credentials.
  For more information, see Section 9.3, "Securing outgoing connections".

8.4. ADDING METADATA TO CONNECTIONS

In a complex topology, it can be useful to add metadata to connections so that messages can be handled programmatically.

Procedure

1. Open the **/etc/qpid-dispatch/qdrouterd.conf** configuration file.
2. Add arbitrary JSON to the **connector** configuration using the 'openProperties' attribute.
   This example adds the property **label** with the value **green**.

```
connector {
   name: broker
   role: route-container
   host: 127.0.0.1
   port: 22180
   saslMechanisms: ANONYMOUS
```

```
    openProperties: {
      "label": "green"
    }
  }
```

Note the following restrictions on the JSON entries:

- ASCII characters only for keys
- The following keys are not allowed:
  - product
  - version
  - failover-server-list
  - network-host
  - port
  - scheme
  - hostname
  - any key starting with **qd.**
  - any key starting with **x-opt-qd.**

The **openProperties** attribute can only be set for a connector with a **normal** or **route-container** role. You cannot set the attribute for connectors that have the following settings:

- **role: inter-router**
- **role: edge**
- **http: true**

The JSON format supports lists, maps and multiple entries, for example:

```
connector {
  name: broker
  role: route-container
  host: 127.0.0.1
  port: 22180
  saslMechanisms: ANONYMOUS
  openProperties: {
    "foo": "bar",
    "integer": 7,
 "list":  ["a", 1, "b", -9, true],
 "map":  {"key1": null, "key2": [1, 2, 3]},
 }
  cost: 10
}
```

8.5. UNDERSTANDING CONNECTION FAILOVER

If a connection between a router and a remote host fails, connection failover enables the connection to be reestablished automatically on an alternate URL.

A router can use connection failover for both incoming and outgoing connections.

Connection failover for outgoing connections

By default, when you configure a **connector** on a router, the router attempts to maintain an open network transport connection to the configured remote host and port. If the connection cannot be established, the router continually retries until the connection is established. If the connection is established and then fails, the router immediately attempts to reestablish the connection.
When the router establishes a connection to a remote host, the client may provide the router with alternate connection information (sometimes called failover lists) that it can use if the connection is lost. In these cases, rather than attempting to reestablish the connection on the same host, the router will also try the alternate hosts.

Connection failover is particularly useful when the router establishes outgoing connections to a cluster of servers providing the same service.

Connection failover for incoming connections

You can configure a **listener** on a router to provide a list of failover URLs to be used as backups. If the connection is lost, the client can use these failover URLs to reestablish the connection to the router.

CHAPTER 9. SECURING NETWORK CONNECTIONS

You can configure AMQ Interconnect to communicate with clients, routers, and brokers in a secure way by authenticating and encrypting the router’s connections. AMQ Interconnect supports the following security protocols:

- SSL/TLS for certificate-based encryption and mutual authentication
- SASL for authentication with mechanisms

You configure SSL/TLS, SASL (or a combination of both) to secure any of the following:

- Secure connections between routers
- Secure incoming client connections
- Secure outgoing connections

9.1. SECURING CONNECTIONS BETWEEN ROUTERS

Connections between interior routers should be secured with SSL/TLS encryption and authentication (also called mutual authentication) to prevent unauthorized routers (or endpoints pretending to be routers) from joining the network.

SSL/TLS mutual authentication requires an X.509 Certificate Authority (CA) with individual certificates generated for each interior router. Connections between the interior routers are encrypted, and the CA authenticates each incoming inter-router connection.

This procedure describes how to secure a connection between two interior routers using SSL/TLS mutual authentication.

Prerequisites

- An X.509 Certificate Authority must exist for the interior routers.
- A security certificate must be generated for each router and be signed by the CA.
- An inter-router connection must exist between the routers.
  For more information, see Section 8.1, “Connecting routers”.

Procedure

1. On the router that establishes the connection, do the following:
   a. Open the **/etc/qpid-dispatch/qdrouterd.conf**.
   b. If the router does not contain an **sslProfile** that defines the private keys and certificates for the inter-router network, then add one.
      This **sslProfile** contains the locations of the private key and certificates that the router uses to authenticate with its peer.

```
sslProfile {
    name: inter-router-tls
    certFile: /etc/pki/tls/certs/tls.crt
    caCertFile: /etc/pki/tls/certs/ca.crt
    privateKeyFile: /etc/pki/tls/private/tls.key
    password: file:/etc/pki/tls/private/password.txt
    ...
}
```

name

A unique name that you can use to refer to this **sslProfile**.

certFile

The absolute path to the file containing the public certificate for this router.

caCertFile

The absolute path to the CA certificate that was used to sign the router's certificate.

privateKeyFile

The absolute path to the file containing the private key for this router's public certificate.

NOTE

Ensure that the **qdrouterd** or root user can access the private key. For example:

```
chmod 0600 /etc/pki/tls/private/tls.key
chown qdrouterd /etc/pki/tls/private/tls.key
```

password

The password to unlock the certificate key. You do not need to specify this if the certificate key does not have a password. By using different prefixes, you can specify the password several different ways depending on your security requirements:

- Specify the absolute path to a file that contains the password. This is the most secure option, because you can set permissions on the file that contains the password. For example:

  ```
  password: file:/etc/qpid-dispatch-certs/inter-router/password.txt
  ```

- Specify an environment variable that stores the password. Use this option with caution, because the environment of other processes is visible on certain platforms. For example:

  ```
  password: env:CERT_PASSWORD
  ```

- Specify the password in clear text. This option is insecure, so it should only be used if security is not a concern. For example:

  ```
  password: pass:mycertpassword
  ```

c. Configure the inter-router **connector** for this connection to use the **sslProfile** that you created.

```
connector {
    host: 192.0.2.1
```

```
    port: 5001
    role: inter-router
    sslProfile: inter-router-tls
    ...
}
```

sslProfile

The name of the **sslProfile** that defines the SSL/TLS private keys and certificates for the inter-router network.

2. On the router that listens for the connection, do the following:

    a. Open the **/etc/qpid-dispatch/qdrouterd.conf**.

    b. If the router does not contain an **sslProfile** that defines the private keys and certificates for the inter-router network, then add one.

    c. Configure the inter-router **listener** for this connection to use SSL/TLS to secure the connection.

```
listener {
    host: 0.0.0.0
    port: 5001
    role: inter-router
    sslProfile: inter_router_tls
    authenticatePeer: yes
    requireSsl: yes
    saslMechanisms: EXTERNAL
    ...
}
```

sslProfile

The name of the **sslProfile** that defines the SSL/TLS private keys and certificates for the inter-router network.

authenticatePeer

Specify **yes** to authenticate the peer interior router's identity.

requireSsl

Specify **yes** to encrypt the connection with SSL/TLS.

saslMechanisms

Specify **EXTERNAL** to enable X.509 client certificate authentication.

9.2. SECURING INCOMING CLIENT CONNECTIONS

You can use SSL/TLS and SASL to provide the appropriate level of security for client traffic into the router network. You can use the following methods to secure incoming connections to a router from AMQP clients, external containers, or edge routers:

- Enable SSL/TLS encryption
- Enable SSL/TLS client authentication
- Enable user name and password authentication

- Integrate with Kerberos

9.2.1. Enabling SSL/TLS encryption

You can use SSL/TLS to encrypt an incoming connection from a client.

Prerequisites

- An X.509 Certificate Authority (CA) must exist for the client connections.
- A security certificate must be generated and signed by the CA.

Procedure

1. Open the **/etc/qpid-dispatch/qdrouterd.conf** configuration file.
2. If the router does not contain an **sslProfile** that defines the private keys and certificates for client connections, then add one.
   This **sslProfile** contains the locations of the private key and certificates that the router should use to encrypt connections from clients.

```
sslProfile {
    name: service-tls
    certFile: /etc/pki/tls/certs/tls.crt
    caCertFile: /etc/pki/tls/certs/ca.crt
    privateKeyFile: /etc/pki/tls/private/tls.key
    password: file:/etc/pki/tls/private/password.txt
    ...
}
```

name
: A unique name that you can use to refer to this **sslProfile**.

certFile
: The absolute path to the file containing the public certificate for this router.

caCertFile
: The absolute path to the CA certificate that was used to sign the router’s certificate.

privateKeyFile
: The absolute path to the file containing the private key for this router’s public certificate.

> NOTE
>
> Ensure that the **qdrouterd** or root user can access the private key. For example:
>
> ```
> chmod 0600 /etc/pki/tls/private/tls.key
> chown qdrouterd /etc/pki/tls/private/tls.key
> ```

password

The password to unlock the certificate key. You do not need to specify this if the certificate key does not have a password. By using different prefixes, you can specify the password several different ways depending on your security requirements:

- Specify the absolute path to a file that contains the password. This is the most secure option, because you can set permissions on the file that contains the password. For example:

  ```
  password: file:/etc/qpid-dispatch-certs/inter-router/password.txt
  ```

- Specify an environment variable that stores the password. Use this option with caution, because the environment of other processes is visible on certain platforms. For example:

  ```
  password: env:CERT_PASSWORD
  ```

- Specify the password in clear text. This option is insecure, so it should only be used if security is not a concern. For example:

  ```
  password: pass:mycertpassword
  ```

3. Configure the **listener** for this connection to use SSL/TLS to encrypt the connection.
   This example configures a **normal** listener to encrypt connections from clients.

   ```
   listener {
       host: 0.0.0.0
       port: 5672
       role: normal
       sslProfile: inter_router_tls
       requireSsl: yes
       ...
   }
   ```

   sslProfile

   The name of the **sslProfile** that defines the SSL/TLS private keys and certificates for client connections.

   requireSsl

   Specify **true** to encrypt the connection with SSL/TLS.

9.2.2. Enabling SSL/TLS client authentication

In addition to SSL/TLS encryption, you can also use SSL/TLS to authenticate an incoming connection from a client. With this method, a clients must present its own X.509 certificate to the router, which the router uses to verify the client’s identity.

Prerequisites

- SSL/TLS encryption must be configured.
  For more information, see Section 9.2.1, “Enabling SSL/TLS encryption” .
- The client must have an X.509 certificate that it can use to authenticate to the router.

Procedure

1. Open the **/etc/qpid-dispatch/qdrouterd.conf** configuration file.

2. Configure the **listener** for this connection to use SSL/TLS to authenticate the client.
   This example adds SSL/TLS authentication to a **normal** listener to authenticate incoming connections from a client. The client will only be able to connect to the router by presenting its own X.509 certificate to the router, which the router will use to verify the client's identity.

   ```
   listener {
       host: 0.0.0.0
       port: 5672
       role: normal
       sslProfile: service-tls
       requireSsl: yes
       authenticatePeer: yes
       saslMechanisms: EXTERNAL
       ...
   }
   ```

   authenticatePeer
   Specify **yes** to authenticate the client's identity.

   saslMechanisms
   Specify **EXTERNAL** to enable X.509 client certificate authentication.

9.2.3. Enabling user name and password authentication

You can use the SASL PLAIN mechanism to authenticate incoming client connections against a set of user names and passwords. You can use this method by itself, or you can combine it with SSL/TLS encryption.

Prerequisites

- The **cyrus-sasl-plain** plugin is installed.
  Cyrus SASL uses plugins to support specific SASL mechanisms. Before you can use a particular SASL mechanism, the relevant plugin must be installed.

  To see a list of Cyrus SASL plugins in Red Hat Enterprise Linux, use the **yum search cyrus-sasl** command. To install a Cyrus SASL plugin, use the **yum install *<plugin>*** command.

Procedure

1. If necessary, add the user names and passwords to the SASL database.
   This example adds a new user (user1@example.com) to the SASL database (qdrouterd.sasldb):

   ```
   $ sudo saslpasswd2 -c -f qdrouterd.sasldb -u example.com user1
   ```

   NOTE

   The full user name is the user name you entered plus the domain name (***<user-name>***@***<domain-name>***). Providing a domain name is not required when you add a user to the database, but if you do not provide one, a default domain will be added automatically (the hostname of the machine on which the tool is running).

2. Ensure that the **qdrouterd** process can read the SASL database.
   If the **qdrouterd** process runs as an unprivileged user, you might need to adjust the permissions or ownership of the SASL database so that the router can read it.

   This example makes the qdrouterd user the owner of the SASL database:

   ```
   $ sudo chown qdrouterd /var/lib/qdrouterd/qdrouterd.sasldb
   ```

3. Open the **/etc/sasl2/qdrouterd.conf** configuration file.
   This example shows a **/etc/sasl2/qdrouterd.conf** configuration file:

   ```
   pwcheck_method: auxprop
   auxprop_plugin: sasldb
   sasldb_path: qdrouterd.sasldb
   mech_list: ANONYMOUS DIGEST-MD5 EXTERNAL PLAIN GSSAPI
   ```

4. Verify that the **mech_list** attribute contains the **PLAIN** mechanism.

5. Open the **/etc/qpid-dispatch/qdrouterd.conf** configuration file.

6. In the **router** section, specify the path to the SASL configuration file.

   ```
   router {
       mode: interior
       id: Router.A
       saslConfigDir: /etc/sasl2/
   }
   ```

   saslConfigDir

   The absolute path to the SASL configuration file that contains the path to the SASL database that stores the user names and passwords.

7. Configure the **listener** for this connection to authenticate clients using SASL PLAIN.
   This example configures basic user name and password authentication for a **listener**. In this case, no SSL/TLS encryption is being used.

   ```
   listener {
       host: 0.0.0.0
       port: 5672
       authenticatePeer: yes
       saslMechanisms: PLAIN
       }
   ```

9.2.4. Integrating with Kerberos

If you have implemented Kerberos in your environment, you can use it with the **GSSAPI** SASL mechanism to authenticate incoming connections.

Prerequisites

- A Kerberos infrastructure must be deployed in your environment.

- In the Kerberos environment, a service principal of **amqp/<hostname>@<realm>** must be configured.
  This is the service principal that AMQ Interconnect uses.

- The **cyrus-sasl-gssapi** package must be installed on each client and the router host machine.

Procedure

1. On the router's host machine, open the **/etc/sasl2/qdrouterd.conf** configuration file.
   This example shows a **/etc/sasl2/qdrouterd.conf** configuration file:

   ```
   pwcheck_method: auxprop
   auxprop_plugin: sasldb
   sasldb_path: qdrouterd.sasldb
   keytab: /etc/krb5.keytab
   mech_list: ANONYMOUS DIGEST-MD5 EXTERNAL PLAIN GSSAPI
   ```

2. Verify the following:

   - The **mech_list** attribute contains the **GSSAPI** mechanism.

   - The **keytab** attribute points to the location of the keytab file.

3. Open the **/etc/qpid-dispatch/qdrouterd.conf** configuration file.

4. In the **router** section, specify the path to the SASL configuration file.

   ```
   router {
       mode: interior
       id: Router.A
       saslConfigDir: /etc/sasl2/
   }
   ```

   saslConfigDir

   The absolute path to the SASL configuration file that contains the path to the SASL database.

5. For each incoming connection using Kerberos for authentication, set the **listener** to use the **GSSAPI** mechanism.

   ```
   listener {
       host: 0.0.0.0
       port: 5672
       authenticatePeer: yes
       saslMechanisms: GSSAPI
       }
   ```

9.3. SECURING OUTGOING CONNECTIONS

If a router is configured to create connections to external AMQP containers (such as message brokers), you can use the following methods to secure the connection:

- Connect using SSL/TLS encryption (one-way authentication)

- Connect using SSL/TLS mutual authentication
- Connect using user name and password authentication (with or without SSL/TLS encryption)

9.3.1. Connecting using one-way SSL/TLS authentication

You can connect to an external AMQP container (such as a broker) using one-way SSL/TLS. With this method, the router validates the external AMQP container's server certificate to verify its identity.

Procedure

1. Open the **/etc/qpid-dispatch/qdrouterd.conf** configuration file.
2. If the router does not contain an **sslProfile** that defines a certificate that can be used to validate the external AMQP container's identity, then add one.

   ```
   sslProfile {
       name: broker-tls
       caCertFile: /etc/qpid-dispatch-certs/ca.crt
       ...
   }
   ```

   name

   A unique name that you can use to refer to this **sslProfile**.

   caCertFile

   The absolute path to the CA certificate used to verify the external AMQP container's identity.

3. Configure the **connector** for this connection to use SSL/TLS to validate the server certificate received by the broker during the SSL handshake.
   This example configures a **connector** to a broker. When the router connects to the broker, it will use the CA certificate defined in the **broker-tls sslProfile** to validate the server certificate received from the broker.

   ```
   connector {
       host: 192.0.2.1
       port: 5672
       role: route-container
       sslProfile: broker-tls
       ...
   }
   ```

   sslProfile

   The name of the **sslProfile** that defines the certificate to use to validate the external AMQP container's identity.

9.3.2. Connecting using mutual SSL/TLS authentication

You can connect to an external AMQP container (such as a broker) using mutual SSL/TLS authentication. With this method, the router, acting as a client, provides a certificate to the external AMQP container so that it can verify the router's identity.

Prerequisites

- An X.509 Certificate Authority (CA) must exist for the router.
- A security certificate must be generated for the router and be signed by the CA.

Procedure

1. Open the **/etc/qpid-dispatch/qdrouterd.conf** configuration file.
2. If the router does not contain an **sslProfile** that defines the private keys and certificates to connect to the external AMQP container, then add one.
   This **sslProfile** contains the locations of the private key and certificates that the router should use to authenticate with its peer.

```
sslProfile {
    name: broker-tls
    certFile: /etc/pki/tls/certs/tls.crt
    caCertFile: /etc/pki/tls/certs/ca.crt
    privateKeyFile: /etc/pki/tls/private/tls.key
    password: file:/etc/pki/tls/private/password.txt
    ...
}
```

name
   A unique name that you can use to refer to this **sslProfile**.

certFile
   The absolute path to the file containing the public certificate for this router.

caCertFile
   The absolute path to the CA certificate that was used to sign the router's certificate.

privateKeyFile
   The absolute path to the file containing the private key for this router's public certificate.

NOTE

Ensure that the **qdrouterd** or root user can access the private key. For example:

```
chmod 0600 /etc/pki/tls/private/tls.key
chown qdrouterd /etc/pki/tls/private/tls.key
```

password
   The password to unlock the certificate key. You do not need to specify this if the certificate key does not have a password. By using different prefixes, you can specify the password several different ways depending on your security requirements:

- Specify the absolute path to a file that contains the password. This is the most secure option, because you can set permissions on the file that contains the password. For example:

```
password: file:/etc/qpid-dispatch-certs/inter-router/password.txt
```

- Specify an environment variable that stores the password. Use this option with caution, because the environment of other processes is visible on certain platforms. For example:

  ```
  password: env:CERT_PASSWORD
  ```

- Specify the password in clear text. This option is insecure, so it should only be used if security is not a concern. For example:

  ```
  password: pass:mycertpassword
  ```

3. Configure the **connector** for this connection to use the **sslProfile** that you created.

   ```
   connector {
       host: 192.0.2.1
       port: 5672
       role: route-container
       sslProfile: broker-tls
       saslMechanisms: EXTERNAL
       ...
   }
   ```

   sslProfile

   The name of the **sslProfile** that defines the SSL/TLS private keys and certificates for the inter-router network.

9.3.3. Connecting using user name and password authentication

You can use the SASL PLAIN mechanism to connect to an external AMQP container that requires a user name and password. You can use this method by itself, or you can combine it with SSL/TLS encryption.

Prerequisites

- The **cyrus-sasl-plain** plugin is installed.
  Cyrus SASL uses plugins to support specific SASL mechanisms. Before you can use a particular SASL mechanism, the relevant plugin must be installed.

  To see a list of Cyrus SASL plugins in Red Hat Enterprise Linux, use the **yum search cyrus-sasl** command. To install a Cyrus SASL plugin, use the **yum install *<plugin>*** command.

Procedure

1. Open the **/etc/qpid-dispatch/qdrouterd.conf** configuration file.

2. Configure the **connector** for this connection to provide user name and password credentials to the external AMQP container.

   ```
   connector {
       host: 192.0.2.1
       port: 5672
       role: route-container
       saslMechanisms: PLAIN
   ```

```
    saslUsername: user
    saslPassword: file:/path/to/file/password.txt
    }
```

saslPassword

The password to connect to the peer. By using different prefixes, you can specify the password several different ways depending on your security requirements:

- Specify the absolute path to a file that contains the password. This is the most secure option, because you can set permissions on the file that contains the password. For example:

  ```
  password: file:/path/to/file/password.txt
  ```

- Specify an environment variable that stores the password. Use this option with caution, because the environment of other processes is visible on certain platforms. For example:

  ```
  password: env:PASSWORD
  ```

- Specify the password in clear text. This option is insecure, so it should only be used if security is not a concern. For example:

  ```
  password: pass:mypassword
  ```

CHAPTER 10. CONFIGURING AUTHORIZATION

You can configure *policies* to secure messaging resources in your messaging environment. Policies ensure that only authorized users can access messaging endpoints through the router network, and that the resources on those endpoints are used in an authorized way.

10.1. TYPES OF POLICIES

AMQ Interconnect provides the following types of policies to control connection and resource limits:

Global policies

Settings for the router. A global policy defines the maximum number of incoming user connections for the router (across all messaging endpoints), and defines how the router should use vhost policies.

Vhost policies

Connection and AMQP resource limits for a router ingress port (called an AMQP virtual host, or vhost). A vhost policy defines what a client using a particular connection can access on any messaging endpoint in the router network.

The resource limits defined in global and vhost policies are applied to user connections only. The limits do not affect inter-router connections or router connections that are outbound to waypoints.

Access to an AMQP resource allowed by policy for a given user connection to a given vhost is granted across the entire router network. Access restrictions are applied only at the router port to which a client is connected and only to resource requests originated by the client.

10.2. HOW POLICIES ENFORCE CONNECTION AND RESOURCE LIMITS

AMQ Interconnect uses policies to determine whether to permit a connection, and if it is permitted, to apply the appropriate resource limits.

When a client creates a connection to a router, the router first determines whether to allow or deny the connection. This decision is based on the following criteria:

- Whether the connection will exceed the router’s global connection limit (defined in the global policy)
- Whether the connection will exceed the vhost’s connection limits (defined in the vhost policy that matches the host to which the connection is directed)

If the connection is allowed, the router assigns the user (the authenticated user name from the connection) to a user group, and enforces the user group’s resource limits for the lifetime of the connection.

10.3. SETTING GLOBAL LIMITS

You can create a global policy to set the incoming connection and message size limits for a router.

Procedure

- In the **/etc/qpid-dispatch/qdrouterd.conf** configuration file, add a **policy** section and set the limits.
  This example sets the incoming connection limit and message size:

  ```
  policy {
      maxConnections: 10000
      maxMessageSize: 2000000
  }
  ```

  maxConnections
  The total number of concurrent client connections that can be open for this router. This limit is always enforced, even if no other policy settings have been defined. The limit is applied to all incoming connections regardless of remote host, authenticated user, or targeted vhost. The default (and the maximum) value is **65535**.

  maxMessageSize
  The maximum size in bytes of AMQP message transfers allowed for this router as messages enter the router network. This limit is applied to transfers over user connections and to transfers to interior routers from edge routers. This limit is not applied to interior-to-interior router connections. This limit may be overridden by vhost or by vhost user group settings. A value of **0** disables this limit. Administrators are advised not set interior router maximum message sizes so low that edge router management requests or responses are blocked. Administrators are also advised to set edge router maximum message sizes lower than the attached interior router maximum message size.

10.4. SETTING CONNECTION AND RESOURCE LIMITS FOR MESSAGING ENDPOINTS

You can define the connection limit and AMQP resource limits for a messaging endpoint by configuring a *vhost policy*. Vhost policies define what resources clients are permitted to access on a messaging endpoint over a particular connection.

NOTE

A vhost is typically the name of the host to which the client connection is directed. For example, if a client application opens a connection to the **amqp://mybroker.example.com:5672/queue01** URL, the vhost would be **mybroker.example.com**.

- Section 10.4.1, “Enabling vhost policies”
- Section 10.4.2, “Creating vhost policies”
- Section 10.4.3, “Creating vhost policies as JSON files”
- Section 10.4.4, “Setting resource limits for outgoing connections”
- Section 10.4.5, “Methods for specifying vhost policy source and target addresses”
- Section 10.4.6, “Vhost policy hostname pattern matching rules”
- Section 10.4.7, “Vhost policy examples”

10.4.1. Enabling vhost policies

You must enable the router to use vhost policies before you can create the policies.

Procedure

- In the **/etc/qpid-dispatch/qdrouterd.conf** configuration file, add a **policy** section if one does not exist, and enable vhost policies for the router.

```
policy {
    ...
    enableVhostPolicy: true
    enableVhostNamePatterns: true
    defaultVhost: $default
}
```

enableVhostPolicy

Enables the router to enforce the connection denials and resource limits defined in the configured vhost policies. The default is **false**, which means that the router will not enforce any vhost policies.

enableVhostNamePatterns

Enables pattern matching for vhost hostnames. If set to **true**, you can use wildcards to specify a range of hostnames for a vhost. If set to **false**, vhost hostnames are treated as literal strings. This means that you must specify the exact hostname for each vhost. The default is **false**.

defaultVhost

The name of the default vhost policy, which is applied to any connection for which a vhost policy has not been configured. The default is **$default**. If **defaultVhost** is not defined, then default vhost processing is disabled.

10.4.2. Creating vhost policies

A vhost policy defines the connection limits and resource limits for users connecting to the router from a remote host. You must create one vhost policy for each remote host.

Prerequisites

Vhost policies must be enabled for the router. For more information, see Section 10.4.1, “Enabling vhost policies”.

Procedure

1. Add a **vhost** section and define the connection and message size limits for the messaging endpoint.
   The connection limits apply to all users that are connected to the vhost. These limits control the number of users that can be connected simultaneously to the vhost.

```
vhost {
    hostname: example.com
    aliases: example.org, example.net
    maxConnections: 10000
    maxMessageSize: 500000
    maxConnectionsPerUser: 100
    maxConnectionsPerHost: 100
    allowUnknownUser: true
    ...
}
```

hostname

The literal hostname of the vhost (the messaging endpoint) or a pattern that matches the vhost hostname. This vhost policy will be applied to any client connection that is directed to the hostname that you specify. This name must be unique; you can only have one vhost policy per hostname.
If **enableVhostNamePatterns** is set to **true**, you can use wildcards to specify a pattern that matches a range of hostnames. For more information, see Section 10.4.6, "Vhost policy hostname pattern matching rules".

aliases

Alternative literal hostnames or patterns that direct the router to use the settings in this vhost. Alias hostnames that match an incoming connection use the settings defined in the vhost section. In a multi-tenant configuration, a connection to a vhost alias uses the base vhost hostname for the tenant namespace. In this example if a connection is directed to vhost **example.org** then the settings from the base vhost hostname **example.com** apply and **example.com** becomes the tenant namespace. Vhost **hostname** and **aliases** settings from all vhosts must be unique.
If **enableVhostNamePatterns** is set to **true**, you can use wildcards to specify a pattern that matches a range of hostname aliases. For more information, see Section 10.4.6, "Vhost policy hostname pattern matching rules".

maxConnections

The global maximum number of concurrent client connections allowed for this vhost. The default is 65535.

maxMessageSize

The maximum size in bytes of AMQP message transfers allowed for connections to this vhost. This limit overrides the policy **maxMessageSize** value and may be overridden by vhost user group settings. A value of **0** disables this limit.

maxConnectionsPerUser

The maximum number of concurrent client connections allowed for any user. The default is 65535.

maxConnectionsPerHost

The maximum number of concurrent client connections allowed for any remote host (the host from which the client is connecting). The default is 65535.

allowUnknownUser

Whether unknown users (users who are not members of a defined user group) are allowed to connect to the vhost. Unknown users are assigned to the **$default** user group and receive **$default** settings. The default is **false**, which means that unknown users are not allowed.

2. In the **vhost** section, beneath the connection settings that you added, add a **groups** entity to define the resource limits.
You define resource limits by user group. A user group specifies the messaging resources the members of the group are allowed to access.

This example shows three user groups: admin, developers, and $default:

```
vhost {
    ...
    groups: {
        admin: {
            users: admin1, admin2
            remoteHosts: 127.0.0.1, ::1
            sources: *
            targets: *
        }
        developers: {
            users: dev1, dev2, dev3
            remoteHosts: *
            sources: myqueue1, myqueue2
            targets: myqueue1, myqueue2
        }
        $default: {
            remoteHosts: *
            allowDynamicSource: true,
            allowAdminStatusUpdate: true,
            sources: myqueue1, myqueue2
            targets: myqueue1, myqueue2
        }
    }
}
```

users

A list of authenticated users for this user group. Use commas to separate multiple users. A user may belong to only one vhost user group.

remoteHosts

A list of remote hosts from which the users may connect. A host can be a hostname, IP address, or IP address range. Use commas to separate multiple hosts. To allow access from all remote hosts, specify a wildcard *****. To deny access from all remote hosts, leave this attribute blank.

maxConnectionsPerUser

The maximum number of connections that may be created by users in this user group. This value, if specified, overrides the vhost **maxConnectionsPerUser** value.

maxConnectionsPerHost

The maximum number of concurrent connections that may be created by users in this user group from any of the permitted remote hosts. This value, if specified, overrides the vhost **maxConnectionsPerUser** value.

maxMessageSize

The maximum size in bytes of AMQP message transfers allowed for connections created by users in this group. This limit overrides the policy and vhost **maxMessageSize** values. A value of **0** disables this limit.

allowDynamicSource

If **true**, connections from users in this group are permitted to attach receivers to dynamic sources. This permits creation of listeners to temporary addresses or temporary queues. If **false**, use of dynamic sources is not permitted.

allowAdminStatusUpdate

If **true**, connections from users in this group are permitted to modify the **adminStatus** of connections. This permits termination of sender or receiver connections. If **false**, the users of this group are prohibited from terminating any connections. Inter-router connections can never be terminated by any usee. The default is **true**, even if the policy is not configured.

allowWaypointLinks

If **true**, connections from users in this group are permitted to attach links using waypoint capabilities. This allows endpoints to act as waypoints (that is, brokers) without the need for configuring auto-links. If **false**, use of waypoint capabilities is not permitted.

allowDynamicLinkRoutes

If **true**, connections from users in this group may dynamically create connection-scoped link route destinations. This allows endpoints to act as link route destinations (that is, brokers) without the need for configuring link routes. If **false**, creation of dynamic link route destinations is not permitted.

allowFallbackLinks

If **true**, connections from users in this group are permitted to attach links using fallback-link capabilities. This allows endpoints to act as fallback destinations (and sources) for addresses that have fallback enabled. If **false**, use of fallback-link capabilities is not permitted.

sources | sourcePattern

A list of AMQP source addresses from which users in this group may receive messages.
Use **sources** to specify one or more literal addresses. To specify multiple addresses, use a comma-separated list. To prevent users in this group from receiving messages from any addresses, leave this attribute blank. To allow access to an address specific to a particular user, specify the **${user}** token. For more information, see Section 10.4.5, “Methods for specifying vhost policy source and target addresses”.

Alternatively, you can use **sourcePattern** to match one or more addresses that correspond to a pattern. A pattern is a sequence of words delimited by either a **.** or / character. You can use wildcard characters to represent a word. The * character matches exactly one word, and the **#** character matches any sequence of zero or more words.

To specify multiple address ranges, use a comma-separated list of address patterns. For more information, see]. To allow access to address ranges that are specific to a particular user, specify the **${user}** token. For more information, see xref:methods-specifying-vhost-policy-source-target-addresses-router-rhel[.

targets | targetPattern

A list of AMQP target addresses from which users in this group may send messages. You can specify multiple AMQP addresses and use user name substitution and address patterns the same way as with source addresses.

3. If necessary, add any advanced user group settings to the vhost user groups.
The advanced user group settings enable you to define resource limits based on the AMQP connection open, session begin, and link attach phases of the connection. For more information, see vhost in the **qdrouterd.conf** man page.

10.4.3. Creating vhost policies as JSON files

As an alternative to using the router configuration file, you can configure vhost policies in JSON files. If you have multiple routers that need to share the same vhost configuration, you can put the vhost configuration JSON files in a location accessible to each router, and then configure the routers to apply the vhost policies defined in these JSON files.

Prerequisites

- Vhost policies must be enabled for the router. For more information, see Section 10.4.1, “Enabling vhost policies”.

Procedure

1. In the **/etc/qpid-dispatch/qdrouterd.conf** configuration file, specify the directory where you want to store the vhost policy definition JSON files.

   ```
   policy {
       ...
       policyDir: /etc/qpid-dispatch-policies
   }
   ```

   policyDir

   The absolute path to the directory that holds vhost policy definition files in JSON format. The router processes all of the vhost policies in each JSON file that is in this directory.

2. In the vhost policy definition directory, create a JSON file for each vhost policy.

   Example 10.1. Vhost Policy Definition JSON File

   ```
   [
       ["vhost", {
           "hostname": "example.com",
           "maxConnections": 10000,
           "maxConnectionsPerUser": 100,
           "maxConnectionsPerHost": 100,
           "allowUnknownUser": true,
           "groups": {
               "admin": {
                   "users": ["admin1", "admin2"],
                   "remoteHosts": ["127.0.0.1", "::1"],
                   "sources": "*",
                   "targets": "*"
               },
               "developers": {
                   "users": ["dev1", "dev2", "dev3"],
                   "remoteHosts": "*",
                   "sources": ["myqueue1", "myqueue2"],
                   "targets": ["myqueue1", "myqueue2"]
               },
               "$default": {
                   "remoteHosts": "*",
                   "allowDynamicSource": true,
                   "sources": ["myqueue1", "myqueue2"],
                   "targets": ["myqueue1", "myqueue2"]
               }
           }
       }]
   ]
   ```

   For more information about these attributes, see Section 10.4.2, “Creating vhost policies”.

10.4.4. Setting resource limits for outgoing connections

If a router establishes an outgoing connection to an external AMQP container (such as a client or broker), you can restrict the resources that the external container can access on the router by configuring a connector vhost policy.

The resource limits that are defined in a connector vhost policy are applied to links that are initiated by the external AMQP container. The connector vhost policy does not restrict links that the router creates.

A connector vhost policy can only be applied to a connector with a **normal** or **route-container** role. You cannot apply connector vhost policies to connectors that have **inter-router** or **edge** roles.

Prerequisites

- Vhost policies are enabled for the router. For more information, see Section 10.4.1, "Enabling vhost policies".

Procedure

1. In the **/etc/qpid-dispatch/qdrouterd.conf** configuration file, add a **vhost** section with a **$connector** user group.

```
vhost {
    hostname: my-connector-policy
    groups: {
        $connector: {
            sources: *
            targets: *
            maxSenders: 5
            maxReceivers: 10
            allowAnonymousSender: true
            allowWaypointLinks: true
        }
    }
}
```

   hostname

   A unique name to identify the connector vhost policy. This name does not represent an actual hostname; therefore, choose a name that will not conflict with an actual vhost hostname.

   $connector

   Identifies this vhost policy as a connector vhost policy. For more information about the resource limits you can apply, see Section 10.4.2, "Creating vhost policies".

2. Apply the connector vhost policy to the connector that establishes the connection to the external AMQP container.
   The following example applies the connector vhost policy that was configured in the previous step:

```
connector {
    host: 192.0.2.10
    port: 5672
    role: normal
    policyVhost: my-connector-policy
}
```

10.4.5. Methods for specifying vhost policy source and target addresses

If you want to allow or deny access to multiple addresses on a vhost, there are several methods you can use to match multiple addresses without having to specify each address individually.

The following table describes the methods a vhost policy can use to specify multiple source and target addresses:

To...	Do this...
Allow all users in the user group to access all source or target addresses	Use a ***** wildcard character. **Example 10.2. Receive from any address** `sources: *`
Prevent all users in the user group from accessing all source or target addresses	Do not specify a value. **Example 10.3. Prohibit message transfers to all addresses** `targets:`

To...	Do this...

Allow access to some resources specific to each user	Use the **${user}** username substitution token. You can use this token with **source**, **target**, **sourcePattern**, and **targetPattern**.
	
	NOTE
	
	You can only specify the **${user}** token once in an AMQP address name or pattern. If there are multiple tokens in an address, only the leftmost token will be substituted.
	
	Example 10.4. Receive from a user-specific address
	
	This definition allows the users in the user group to receive messages from any address that meets any of the following rules:
	● Starts with the prefix **tmp_** and ends with the user name
	● Starts with the prefix **temp** followed by any additional characters
	● Starts with the user name, is followed by **-home-**, and ends with any additional characters
	
	`sources: tmp_${user}, temp*, ${user}-home-*`
	
	Example 10.5. User-specific address patterns
	
	This definition allows the users in the user group to receive messages from any address that meets any of the following rules:
	● Starts with the prefix **tmp** and ends with the user name
	● Starts with the prefix **temp** followed by zero or more additional characters
	● Starts with the user name, is followed by **home**, and ends with one or more additional characters
	
	`sourcePattern: tmp.${user}, temp/#, ${user}.home/*`
	
	NOTE
	
	In an address pattern (**sourcePattern** or **targetPattern**), the username substitution token must be either the first or last token in the pattern. The token must also be alone within its delimited field, which means that it cannot be concatenated with literal text prefixes or suffixes.

10.4.6. Vhost policy hostname pattern matching rules

In a vhost policy, vhost hostnames can be either literal hostnames or patterns that cover a range of hostnames.

A hostname pattern is a sequence of words with one or more of the following wildcard characters:

- * represents exactly one word
- # represents zero or more words

The following table shows some examples of hostname patterns:

This pattern...	Matches...	But not...
***.example.com**	**www.example.com**	**example.comsrv2.www.example.com**
#.example.com	**example.comwww.example.coma.b.c.d.example.com**	**myhost.com**
www.*.test.example.com	**www.a.test.example.com**	**www.test.example.comwww.a.b.c.test.example.com**
www.#.test.example.com	**www.test.example.comwww.a.test.example.comwww.a.b.c.test.example.com**	**test.example.com**

Vhost hostname pattern matching applies the following precedence rules:

Policy pattern	Precedence
Exact match	High
*	Medium
#	Low

NOTE

AMQ Interconnect does not permit you to create vhost hostname patterns that conflict with existing patterns. This includes patterns that can be reduced to be the same as an existing pattern. For example, you would not be able to create the **#.#.#.#.com** pattern if **#.com** already exists.

10.4.7. Vhost policy examples

These examples demonstrate how to use vhost policies to authorize access to messaging resources.

Example 10.6. Defining basic resource limits for a messaging endpoint

In this example, a vhost policy defines resource limits for clients connecting to the **example.com** host.

```
[
    ["vhost", {
        "hostname": "example.com",  1
        "maxConnectionsPerUser": 10,  2
        "allowUnknownUser": true,  3
        "groups": {
            "admin": {
                "users": ["admin1", "admin2"],  4
                "remoteHosts": ["127.0.0.1", "::1"],  5
                "sources": "*",  6
                "targets": "*"  7
            },
            "$default": {
                "remoteHosts": "*",  8
                "sources": ["news*", "sports*" "chat*"],  9
                "targets": "chat*"  10
            }
        }
    }]
]
```

1 The rules defined in this vhost policy will be applied to any user connecting to **example.com**.

2 Each user can open up to 10 connections to the vhost.

3 Any user can connect to this vhost. Users that are not part of the **admin** group are assigned to the **$default** group.

4 If the **admin1** or **admin2** user connects to the vhost, they are assigned to the **admin** user group.

5 Users in the **admin** user group must connect from localhost. If the admin user attempts to connect from any other host, the connection will be denied.

6 Users in the admin user group can receive from any address.

7 Users in the admin user group can send to any address.

8 Any non-admin user is permitted to connect from any host.

9 Non-admin users are permitted to receive messages from any addresses that start with the **news**, **sports**, or **chat** prefixes.

10 Non-admin users are permitted to send messages to any addresses that start with the **chat** prefix.

Example 10.7. Limiting memory consumption

By using the advanced vhost policy attributes, you can control how much system buffer memory a user connection can potentially consume.

In this example, a stock trading site provides services for stock traders. However, the site must also accept high-capacity, automated data feeds from stock exchanges. To prevent trading activity from consuming memory needed for the feeds, a larger amount of system buffer memory is allotted to the feeds than to the traders.

This example uses the **maxSessions** and **maxSessionWindow** attributes to set the buffer memory consumption limits for each AMQP session. These settings are passed directly to the AMQP connection and session negotiations, and do not require any processing cycles on the router.

This example does not show the vhost policy settings that are unrelated to buffer allocation.

```
[
   ["vhost", {
      "hostname": "traders.com",  1
      "groups": {
         "traders": {
            "users": ["trader1", "trader2"],  2
            "maxFrameSize": 10000,
            "maxSessionWindow": 5000000,  3
            "maxSessions": 1  4
         },
         "feeds": {
            "users": ["nyse-feed", "nasdaq-feed"],  5
            "maxFrameSize": 60000,
            "maxSessionWindow": 1200000000,  6
            "maxSessions": 3  7
         }
      }
   }]
]
```

1. The rules defined in this vhost policy will be applied to any user connecting to **traders.com**.
2. The **traders** group includes **trader1**, **trader2**, and any other user defined in the list.
3. At most, 5,000,000 bytes of data can be in flight on each session.
4. Only one session per connection is allowed.
5. The **feeds** group includes two users.
6. At most, 1,200,000,000 bytes of data can be in flight on each session.
7. Up to three sessions per connection are allowed.

CHAPTER 11. CONFIGURING LOGGING

AMQ Interconnect contains internal logging modules that provide important information about each router. For each module, you can configure the logging level, the format of the log file, and the location to which the logs should be written.

11.1. LOGGING MODULES

AMQ Interconnect logs are broken into different categories called *logging modules*. Each module provides important information about a particular aspect of AMQ Interconnect.

DEFAULT

The default module. This module applies defaults to all of the other logging modules.

ROUTER

This module provides information and statistics about the local router. This includes how the router connects to other routers in the network, and information about the remote destinations that are directly reachable from the router (link routes, waypoints, autolinks, and so on).

ROUTER_HELLO

This module provides information about the *Hello* protocol used by interior routers to exchange Hello messages, which include information about the router's ID and a list of its reachable neighbors (the other routers with which this router has bidirectional connectivity).

ROUTER_LS

This module provides information about link-state data between routers, including Router Advertisement (RA), Link State Request (LSR), and Link State Update (LSU) messages.

Periodically, each router sends an LSR to the other routers and receives an LSU with the requested information. Exchanging the above information, each router can compute the next hops in the topology, and the related costs.

ROUTER_MA

This module provides information about the exchange of mobile address information between routers, including Mobile Address Request (MAR) and Mobile Address Update (MAU) messages exchanged between routers. You can use this log to monitor the state of mobile addresses attached to each router.

MESSAGE

This module provides information about AMQP messages sent and received by the router, including information about the address, body, and link. You can use this log to find high-level information about messages on a particular router.

SERVER

This module provides information about how the router is listening for and connecting to other containers in the network (such as clients, routers, and brokers). This information includes the state of AMQP messages sent and received by the broker (open, begin, attach, transfer, flow, and so on), and the related content of those messages.

AGENT

This module provides information about configuration changes made to the router from either editing the router's configuration file or using **qdmanage**.

CONTAINER

This module provides information about the nodes related to the router. This includes only the AMQP relay node.

ERROR

This module provides detailed information about error conditions encountered during execution.

POLICY

This module provides information about policies that have been configured for the router.

Additional resources

- For examples of these logging modules, see Section 16.2, “Troubleshooting using logs”.

11.2. CONFIGURING DEFAULT LOGGING

You can specify the types of events that should be logged, the format of the log entries, and where those entries should be sent.

Procedure

1. In the **/etc/qpid-dispatch/qdrouterd.conf** configuration file, add a **log** section to set the default logging properties:
   This example configures all logging modules to log events starting at the **info** level:

   ```
   log {
       module: DEFAULT
       enable: info+
       includeTimestamp: yes
   }
   ```

   module

   Specify **DEFAULT**.

   enable

   The logging level. You can specify any of the following levels (from lowest to highest):

   - **trace** - provides the most information, but significantly affects system performance
   - **debug** - useful for debugging, but affects system performance
   - **info** - provides general information without affecting system performance
   - **notice** - provides general information, but is less verbose than **info**
   - **warning** - provides information about issues you should be aware of, but which are not errors
   - **error** - error conditions that you should address
   - **critical** - critical system issues that you must address immediately

   To specify multiple levels, use a comma-separated list. You can also use **+** to specify a level and all levels above it. For example, **trace,debug,warning+** enables trace, debug, warning, error, and critical levels. For default logging, you should typically use the **info+** or **notice+** level. These levels will provide general information, warnings, and errors for all modules without affecting the performance of AMQ Interconnect.

   includeTimestamp

   Set this to **yes** to include the timestamp in all logs.

For information about additional log attributes, see log in the **qdrouterd.conf** man page.

2. If you want to configure non-default logging for any of the logging modules, add an additional **log** section for each module that should not follow the default.
   This example configures the **ROUTER** logging module to log **debug** events:

```
log {
    module: ROUTER
    enable: debug
    includeTimestamp: yes
}
```

Additional resources

- For more information about viewing and using logs, see Chapter 16, *Troubleshooting AMQ Interconnect*.

CHAPTER 12. CONFIGURING ROUTING

Routing is the process by which messages are delivered to their destinations. To accomplish this, AMQ Interconnect provides two routing mechanisms: *message routing* and *link routing*.

Message routing

Message routing is the default routing mechanism. You can use it to route messages on a per-message basis between clients directly (direct-routed messaging), or to and from broker queues (brokered messaging).

Link routing

A link route represents a private messaging path between a sender and a receiver in which the router passes the messages between end points. You can use it to connect a client to a service (such as a broker queue).

12.1. CONFIGURING MESSAGE ROUTING

Message routing is the default routing mechanism. You can use it to route messages on a per-message basis between clients directly (direct-routed messaging), or to and from broker queues (brokered messaging).

With message routing, you can do the following:

- Understand message routing concepts
- Configure address semantics (route messages between clients)
- Configure addresses for prioritized message delivery
- Configure brokered messaging
- Understand address pattern matching

12.1.1. Understanding message routing

With message routing, routing is performed on messages as producers send them to a router. When a message arrives on a router, the router routes the message and its *settlement* based on the message's *address* and *routing pattern*.

12.1.1.1. Message routing flow control

AMQ Interconnect uses a *credit-based* flow control mechanism to ensure that producers can only send messages to a router if at least one consumer is available to receive them. Because AMQ Interconnect does not store messages, this credit-based flow control prevents producers from sending messages when there are no consumers present.

A client wishing to send a message to the router must wait until the router has provided it with credit. Attempting to publish a message without credit available will cause the client to block. Once credit is made available, the client will unblock, and the message will be sent to the router.

NOTE

Most AMQP client libraries enable you to determine the amount of credit available to a producer. For more information, consult your client's documentation.

12.1.1.2. Addresses

Addresses determine how messages flow through your router network. An address designates an endpoint in your messaging network, such as:

- Endpoint processes that consume data or offer a service
- Topics that match multiple consumers to multiple producers
- Entities within a messaging broker:
  - Queues
  - Durable Topics
  - Exchanges

When a router receives a message, it uses the message's address to determine where to send the message (either its destination or one step closer to its destination).

AMQ Interconnect considers addresses to be *mobile* in that any user of an address may be directly connected to any router in the router network and may even move around the topology. In cases where messages are broadcast to or balanced across multiple consumers, the users of the address may be connected to multiple routers in the network.

Mobile addresses may be discovered during normal router operation or configured through management settings.

12.1.1.3. Routing patterns

Routing patterns define the paths that a message with a mobile address can take across a network. These routing patterns can be used for both direct routing, in which the router distributes messages between clients without a broker, and indirect routing, in which the router enables clients to exchange messages through a broker.

Routing patterns fall into two categories: Anycast (Balanced and Closest) and Multicast. There is no concept of "unicast" in which there is only one consumer for an address.

Anycast distribution delivers each message to one consumer whereas multicast distribution delivers each message to all consumers.

Each address has one of the following routing patterns, which define the path that a message with the address can take across the messaging network:

Balanced

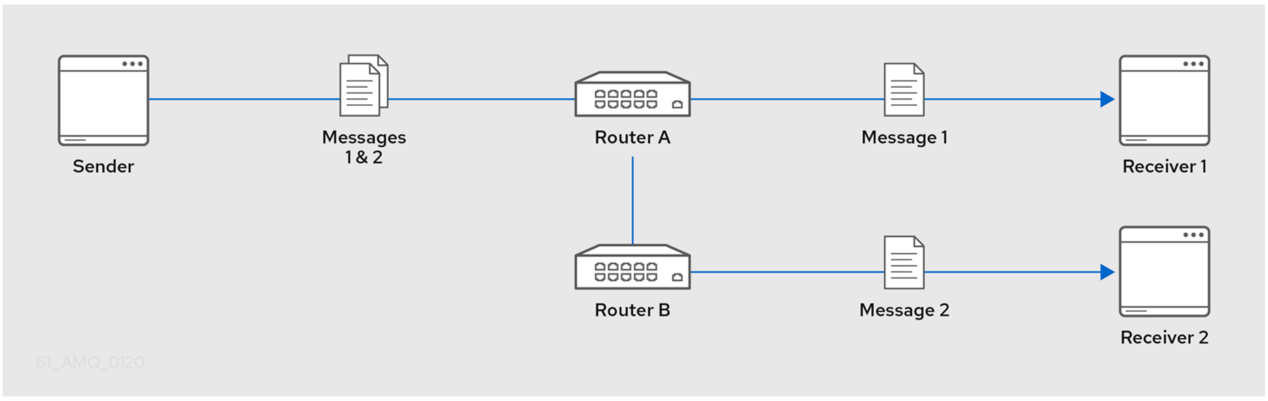
An anycast method that allows multiple consumers to use the same address. Each message is delivered to a single consumer only, and AMQ Interconnect attempts to balance the traffic load across the router network.
If multiple consumers are attached to the same address, each router determines which outbound path should receive a message by considering each path's current number of unsettled deliveries. This means that more messages will be delivered along paths where deliveries are settled at higher rates.

NOTE

AMQ Interconnect neither measures nor uses message settlement time to determine which outbound path to use.

In this scenario, the messages are spread across both receivers regardless of path length:

Figure 12.1. Balanced Message Routing

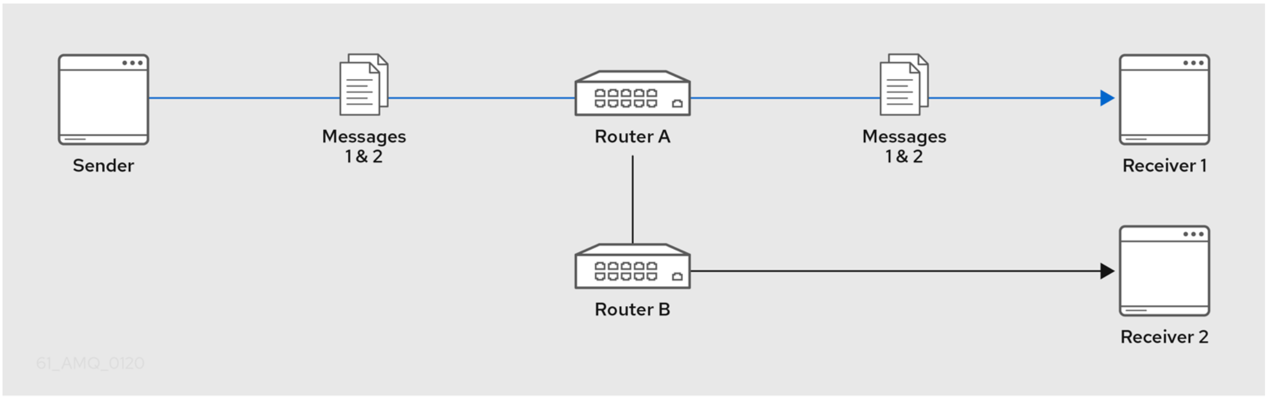


Closest

An anycast method in which every message is sent along the shortest path to reach the destination, even if there are other consumers for the same address.
AMQ Interconnect determines the shortest path based on the topology cost to reach each of the consumers. If there are multiple consumers with the same lowest cost, messages will be spread evenly among those consumers.

In this scenario, all messages sent by **Sender** will be delivered to **Receiver 1**:

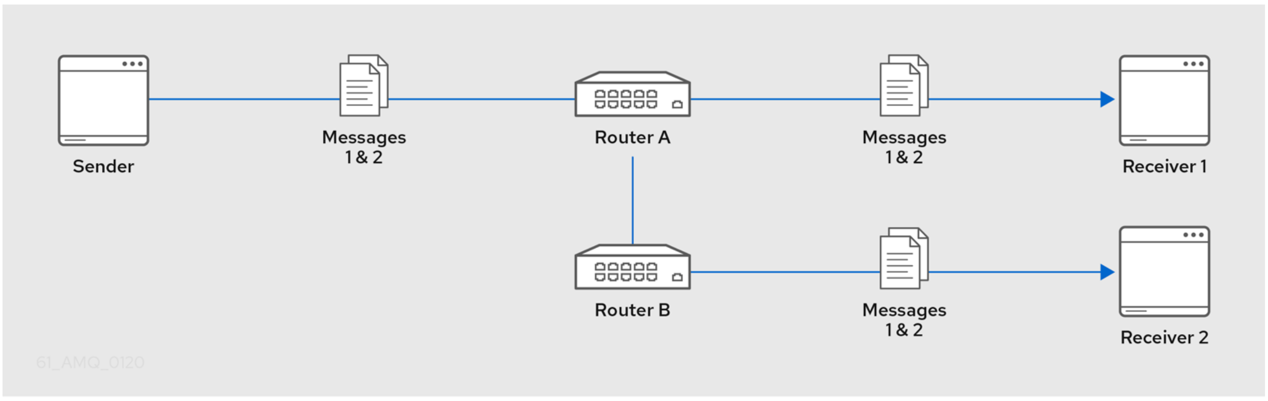
Figure 12.2. Closest Message Routing



Multicast

Messages are sent to all consumers attached to the address. Each consumer will receive one copy of the message.
In this scenario, all messages are sent to all receivers:

Figure 12.3. Multicast Message Routing



12.1.1.4. Message settlement and reliability

AMQ Interconnect can deliver messages with the following degrees of reliability:

- At most once
- At least once
- Exactly once

The level of reliability is negotiated between the producer and the router when the producer establishes a link to the router. To achieve the negotiated level of reliability, AMQ Interconnect treats all messages as either *pre-settled* or *unsettled*.

Pre-settled

Sometimes called *fire and forget*, the router settles the incoming and outgoing deliveries and propagates the settlement to the message's destination. However, it does not guarantee delivery.

Unsettled

AMQ Interconnect propagates the settlement between the producer and consumer. For an anycast address, the router associates the incoming delivery with the resulting outgoing delivery. Based on this association, the router propagates changes in delivery state from the consumer to the producer. For a multicast address, the router associates the incoming delivery with all outbound deliveries. The router waits for each consumer to set their delivery's final state. After all outgoing deliveries have reached their final state, the router sets a final delivery state for the original inbound delivery and passes it to the producer.

The following table describes the reliability guarantees for unsettled messages sent to an anycast or multicast address:

Final disposition	Anycast	Multicast
accepted	The consumer accepted the message.	At least one consumer accepted the message, but no consumers rejected it.
released	The message did not reach its destination.	The message did not reach any of the consumers.

Final disposition	Anycast	Multicast
modified	The message may or may not have reached its destination. The delivery is considered to be "in-doubt" and should be re-sent if "at least once" delivery is required.	The message may or may not have reached any of the consumers. However, no consumers rejected or accepted it.
rejected	The consumer rejected the message.	At least one consumer rejected the message.

12.1.2. Configuring address semantics

You can route messages between clients without using a broker. In a brokerless scenario (sometimes called *direct-routed messaging*), AMQ Interconnect routes messages between clients directly.

To route messages between clients, you configure an address with a routing distribution pattern. When a router receives a message with this address, the message is routed to its destination or destinations based on the address's routing distribution pattern.

Procedure

1. In the **/etc/qpid-dispatch/qdrouterd.conf** configuration file, add an **address** section.

```
address {
    prefix: my_address
    distribution: multicast
    ...
}
```

prefix | pattern

The address or group of addresses to which the address settings should be applied. You can specify a prefix to match an exact address or beginning segment of an address. Alternatively, you can specify a pattern to match an address using wildcards.
A *prefix* matches either an exact address or the beginning segment within an address that is delimited by either a **.** or / character. For example, the prefix **my_address** would match the address **my_address** as well as **my_address.1** and **my_address/1**. However, it would not match **my_address1**.

A *pattern* matches an address that corresponds to a pattern. A pattern is a sequence of words delimited by either a **.** or / character. You can use wildcard characters to represent a word. The ***** character matches exactly one word, and the **#** character matches any sequence of zero or more words.

The ***** and **#** characters are reserved as wildcards. Therefore, you should not use them in the message address.

For more information about creating address patterns, see Section 12.1.5, “Address pattern matching”.

NOTE

You can convert a **prefix** value to a **pattern** by appending **/#** to it. For example, the prefix **a/b/c** is equivalent to the pattern **a/b/c/#**.

distribution

The message distribution pattern. The default is **balanced**, but you can specify any of the following options:

- **balanced** - Messages sent to the address will be routed to one of the receivers, and the routing network will attempt to balance the traffic load based on the rate of settlement.
- **closest** - Messages sent to the address are sent on the shortest path to reach the destination. It means that if there are multiple receivers for the same address, only the closest one will receive the message.
- **multicast** - Messages are sent to all receivers that are attached to the address in a *publish/subscribe* model.
  For more information about message distribution patterns, see Section 12.1.1.3, "Routing patterns".

For information about additional attributes, see address in the **qdrouterd.conf** man page.

2. Add the same **address** section to any other routers that need to use the address.
   The **address** that you added to this router configuration file only controls how this router distributes messages sent to the address. If you have additional routers in your router network that should distribute messages for this address, then you must add the same **address** section to each of their configuration files.

12.1.3. Configuring addresses for prioritized message delivery

You can set the priority level of an address to control how AMQ Interconnect processes messages sent to that address. Within the scope of a connection, AMQ Interconnect attempts to process messages based on their priority. For a connection with a large volume of messages in flight, this lowers the latency for higher-priority messages.

Assigning a high priority level to an address does not guarantee that messages sent to the address will be delivered before messages sent to lower-priority addresses. However, higher-priority messages will travel more quickly through the router network than they otherwise would.

NOTE

You can also control the priority level of individual messages by setting the priority level in the message header. However, the address priority takes precedence: if you send a prioritized message to an address with a different priority level, the router will use the address priority level.

Procedure

- In the **/etc/qpid-dispatch/qdrouterd.conf** configuration file, add or edit an address and assign a priority level.
  This example adds an address with the highest priority level. The router will attempt to deliver messages sent to this address before messages with lower priority levels.

```
address {
    prefix: my-high-priority-address
    priority: 9
    ...
}
```

priority

The priority level to assign to all messages sent to this address. The range of valid priority levels is 0-9, in which the higher the number, the higher the priority. The default is 4.

Additional resources

- For more information about setting the priority level in a message, see the AMQP 1.0 specification.

12.1.4. Configuring brokered messaging

If you require "store and forward" capabilities, you can configure AMQ Interconnect to use brokered messaging. In this scenario, clients connect to a router to send and receive messages, and the router routes the messages to or from queues on a message broker.

You can configure the following:

- Route messages through broker queues
  You can route messages to a queue hosted on a single broker, or route messages to a *sharded queue* distributed across multiple brokers.
- Store and retrieve undeliverable messages on a broker queue

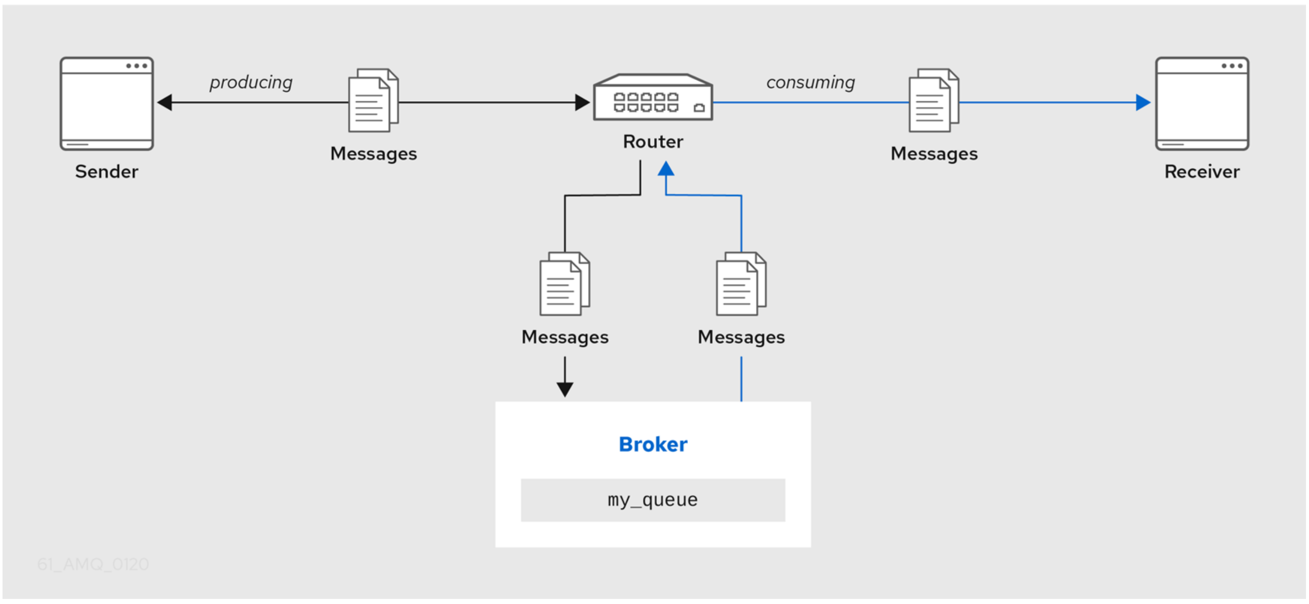
12.1.4.1. How AMQ Interconnect enables brokered messaging

Brokered messaging enables AMQ Interconnect to store messages on a broker queue. This requires a connection to the broker, a *waypoint* address to represent the broker queue, and *autolinks* to attach to the waypoint address.

An autolink is a link that is automatically created by the router to attach to a waypoint address. With autolinks, client traffic is handled on the router, not the broker. Clients attach their links to the router, and then the router uses internal autolinks to connect to the queue on the broker. Therefore, the queue will always have a single producer and a single consumer regardless of how many clients are attached to the router.

Using autolinks is a form of *message routing*, as distinct from *link routing*. It is recommended to use link routing if you want to use semantics associated with a consumer, for example, the **undeliverable-here=true** modified delivery state.

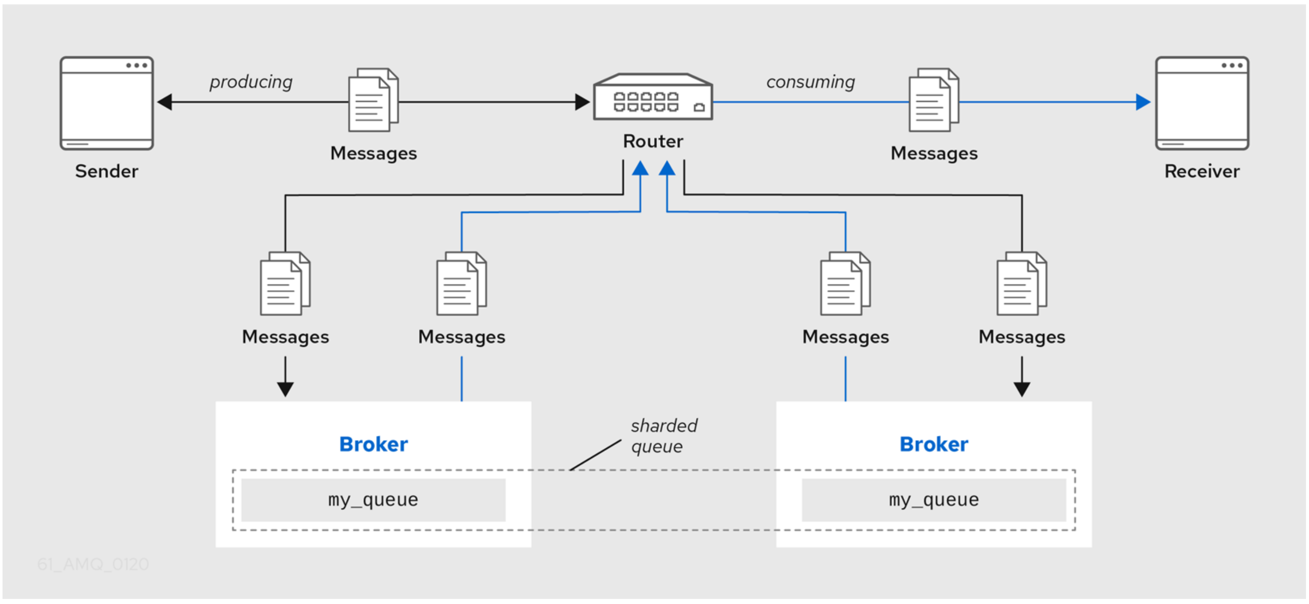
Figure 12.4. Brokered messaging



In this diagram, the sender connects to the router and sends messages to my_queue. The router attaches an outgoing link to the broker, and then sends the messages to my_queue. Later, the receiver connects to the router and requests messages from my_queue. The router attaches an incoming link to the broker to receive the messages from my_queue, and then delivers them to the receiver.

You can also route messages to a *sharded queue*, which is a single, logical queue comprised of multiple, underlying physical queues. Using queue sharding, it is possible to distribute a single queue over multiple brokers. Clients can connect to any of the brokers that hold a shard to send and receive messages.

Figure 12.5. Brokered messaging with sharded queue



In this diagram, a sharded queue (my_queue) is distributed across two brokers. The router is connected to the clients and to both brokers. The sender connects to the router and sends messages to my_queue. The router attaches an outgoing link to each broker, and then sends messages to each shard (by default, the routing distribution is **balanced**). Later, the receiver connects to the router and requests all of the messages from my_queue. The router attaches an incoming link to one of the brokers to receive the messages from my_queue, and then delivers them to the receiver.

12.1.4.2. Routing messages through broker queues

You can route messages to and from a broker queue to provide clients with access to the queue through a router. In this scenario, clients connect to a router to send and receive messages, and the router routes the messages to or from the broker queue.

You can route messages to a queue hosted on a single broker, or route messages to a *sharded queue* distributed across multiple brokers.

Procedure

1. In the **/etc/qpid-dispatch/qdrouterd.conf** configuration file, add a waypoint address for the broker queue.
   A waypoint address identifies a queue on a broker to which you want to route messages. This example adds a waypoint address for the **my_queue** queue:

   ```
   address {
       prefix: my_queue
       waypoint: yes
   }
   ```

   prefix | pattern

   The address prefix or pattern that matches the broker queue to which you want to send messages. You can specify a prefix to match an exact address or beginning segment of an address. Alternatively, you can specify a pattern to match an address using wildcards.
   A *prefix* matches either an exact address or the beginning segment within an address that is delimited by either a **.** or / character. For example, the prefix **my_address** would match the address **my_address** as well as **my_address.1** and **my_address/1**. However, it would not match **my_address1**.

   A *pattern* matches an address that corresponds to a pattern. A pattern is a sequence of words delimited by either a **.** or / character. You can use wildcard characters to represent a word. The ***** character matches exactly one word, and the **#** character matches any sequence of zero or more words.

   The ***** and **#** characters are reserved as wildcards. Therefore, you should not use them in the message address.

   For more information about creating address patterns, see Section 12.1.5, “Address pattern matching”.

   NOTE

   You can convert a **prefix** value to a **pattern** by appending **/#** to it. For example, the prefix **a/b/c** is equivalent to the pattern **a/b/c/#**.

   waypoint

   Set this attribute to **yes** so that the router handles messages sent to this address as a waypoint.

2. Connect the router to the broker.

   a. Add an outgoing connection to the broker if one does not exist.
      If the queue is sharded across multiple brokers, you must add a connection for each broker. For more information, see Section 8.3, “Connecting to external AMQP containers” .

NOTE

If the connection to the broker fails, AMQ Interconnect automatically attempts to reestablish the connection and reroute message deliveries to any available alternate destinations. However, some deliveries could be returned to the sender with a **RELEASED** or **MODIFIED** disposition. Therefore, you should ensure that your clients can handle these deliveries appropriately (generally by resending them).

b. If you want to send messages to the broker queue, add an *outgoing* autolink to the broker queue.
   If the queue is sharded across multiple brokers, you must add an outgoing autolink for each broker.

   This example configures an outgoing auto link to send messages to a broker queue:

```
autoLink {
    address: my_queue
    connection: my_broker
    direction: out
    ...
}
```

   address
      The address of the broker queue. When the autolink is created, it will be attached to this address.

   externalAddress
      An optional alternate address for the broker queue. You use an external address if the broker queue should have a different address than that which the sender uses. In this scenario, senders send messages to the **address** address, and then the router routes them to the broker queue represented by the **externalAddress** address.

   connection | containerID
      How the router should connect to the broker. You can specify either an outgoing connection (**connection**) or the container ID of the broker (**containerID**).

   direction
      Set this attribute to **out** to specify that this autolink can send messages from the router to the broker.

   For information about additional attributes, see autoLink in the **qdrouterd.conf** man page.

3. If you want to receive messages from the broker queue, add an *incoming* autolink from the broker queue:
   If the queue is sharded across multiple brokers, you must add an outgoing autolink for each broker.

   This example configures an incoming auto link to receive messages from a broker queue:

```
autoLink {
    address: my_queue
    connection: my_broker
    direction: in
    ...
}
```

–

address

The address of the broker queue. When the autolink is created, it will be attached to this address.

externalAddress

An optional alternate address for the broker queue. You use an external address if the broker queue should have a different address than that which the receiver uses. In this scenario, receivers receive messages from the **address** address, and the router retrieves them from the broker queue represented by the **externalAddress** address.

connection | containerID

How the router should connect to the broker. You can specify either an outgoing connection (**connection**) or the container ID of the broker (**containerID**).

direction

Set this attribute to **in** to specify that this autolink can receive messages from the broker to the router.

For information about additional attributes, see autoLink in the **qdrouterd.conf** man page.

12.1.4.3. Handling undeliverable messages

You handle undeliverable messages for an address by configuring autolinks that point to *fallback destinations*. A fallback destination (such as a queue on a broker) stores messages that are not directly routable to any consumers.

During normal message delivery, AMQ Interconnect delivers messages to the consumers that are attached to the router network. However, if no consumers are reachable, the messages are diverted to any fallback destinations that were configured for the address (if the autolinks that point to the fallback destinations are active). When a consumer reconnects and becomes reachable again, it receives the messages stored at the fallback destination.

NOTE

AMQ Interconnect preserves the original delivery order for messages stored at a fallback destination. However, when a consumer reconnects, any new messages produced while the queue is draining will be interleaved with the messages stored at the fallback destination.

Prerequisites

- The router is connected to a broker.
  For more information, see Section 8.3, “Connecting to external AMQP containers” .

Procedure

This procedure enables fallback for an address and configures autolinks to connect to the broker queue that provides the fallback destination for the address.

1. In the **/etc/qpid-dispatch/qdrouterd.conf** configuration file, enable fallback destinations for the address.

```
address {
  prefix: my_address
  enableFallback: yes
}
```

2. Add an *outgoing* autolink to a queue on the broker.
   For the address for which you enabled fallback, if messages are not routable to any consumers, the router will use this autolink to send the messages to a queue on the broker.

   ```
   autoLink {
       address: my_address.2
       direction: out
       connection: my_broker
       fallback: yes
   }
   ```

3. If you want the router to send queued messages to attached consumers as soon as they connect to the router network, add an *incoming* autolink.
   As soon as a consumer attaches to the router, it will receive the messages stored in the broker queue, along with any new messages sent by the producer. The original delivery order of the queued messages is preserved; however, the queued messages will be interleaved with the new messages.

   If you do not add the incoming autolink, the messages will be stored on the broker, but will not be sent to consumers when they attach to the router.

   ```
   autoLink {
       address: my_address.2
       direction: in
       connection: my_broker
       fallback: yes
   }
   ```

12.1.5. Address pattern matching

In some router configuration scenarios, you might need to use pattern matching to match a range of addresses rather than a single, literal address. Address patterns match any address that corresponds to the pattern.

An address pattern is a sequence of tokens (typically words) that are delimited by either **.** or / characters. They also can contain special wildcard characters that represent words:

- * represents exactly one word
- **#** represents zero or more words

Example 12.1. Address pattern

This address contains two tokens, separated by the / delimiter:

my/address

Example 12.2. Address pattern with wildcard

This address contains three tokens. The * is a wildcard, representing any single word that might be between **my** and **address**:

```
my/*/address
```

The following table shows some address patterns and examples of the addresses that would match them:

This pattern...	Matches...	But not...
news/*	**news/europe** **news/usa**	**news** **news/usa/sports**
news/#	**news** **news/europe** **news/usa/sports**	**europe** **usa**
news/europe/#	**news/europe** **news/europe/sports** **news/europe/politics/fr**	**news/usa** **europe**
news/*/sports	**news/europe/sports** **news/usa/sports**	**news** **news/europe/fr/sports**

12.2. CREATING LINK ROUTES

A link route represents a private messaging path between a sender and a receiver in which the router passes the messages between end points. You can use it to connect a client to a service (such as a broker queue).

12.2.1. Understanding link routing

Link routing provides an alternative strategy for brokered messaging. A link route represents a private messaging path between a sender and a receiver in which the router passes the messages between end points. You can think of a link route as a "virtual connection" or "tunnel" that travels from a sender, through the router network, to a receiver.

With link routing, routing is performed on link-attach frames, which are chained together to form a virtual messaging path that directly connects a sender and receiver. Once a link route is established, the transfer of message deliveries, flow frames, and dispositions is performed across the link route.

12.2.1.1. Link routing flow control

Unlike message routing, with link routing, the sender and receiver handle flow control directly: the receiver grants link credits, which is the number of messages it is able to receive. The router sends them directly to the sender, and then the sender sends the messages based on the credits that the receiver granted.

12.2.1.2. Link route addresses

A link route address represents a broker queue, topic, or other service. When a client attaches a link route address to a router, the router propagates a link attachment to the broker resource identified by the address.

Using link route addresses, the router network does not participate in aggregated message distribution. The router simply passes message delivery and settlement between the two end points.

12.2.1.3. Routing patterns for link routing

Routing patterns are not used with link routing, because there is a direct link between the sender and receiver. The router only makes a routing decision when it receives the initial link-attach request frame. Once the link is established, the router passes the messages along the link in a balanced distribution.

12.2.2. Creating a link route

Link routes establish a link between a sender and a receiver that travels through a router. You can configure inward and outward link routes to enable the router to receive link-attaches from clients and to send them to a particular destination.

With link routing, client traffic is handled on the broker, not the router. Clients have a direct link through the router to a broker's queue. Therefore, each client is a separate producer or consumer.

NOTE

If the connection to the broker fails, the routed links are detached, and the router will attempt to reconnect to the broker (or its backup). Once the connection is reestablished, the link route to the broker will become reachable again.

From the client's perspective, the client will see the detached links (that is, the senders or receivers), but not the failed connection. Therefore, if you want the client to reattach dropped links in the event of a broker connection failure, you must configure this functionality on the client. Alternatively, you can use message routing with autolinks instead of link routing. For more information, see Section 12.1.4.2, "Routing messages through broker queues".

Procedure

1. Add an outgoing connection to the broker if one does not exist.
   If the queue is sharded across multiple brokers, you must add a connection for each broker. For more information, see Section 8.3, "Connecting to external AMQP containers" .

2. If you want clients to send local transactions to the broker, create a link route for the transaction coordinator:

```
linkRoute {
    prefix: $coordinator (1)
    connection: my_broker
    direction: in
}
```

(1) The **$coordinator** prefix designates this link route as a transaction coordinator. When the client opens a transacted session, the requests to start and end the transaction are propagated along this link route to the broker.

AMQ Interconnect does not support routing transactions to multiple brokers. If you have multiple brokers in your environment, choose a single broker and route all transactions to it.

3. If you want clients to send messages on this link route, create an incoming link route:

```
linkRoute {
    prefix: my_queue
    connection: my_broker
    direction: in
    ...
}
```

prefix | pattern

The address prefix or pattern that matches the broker queue that should be the destination for routed link-attaches. All messages that match this prefix or pattern will be distributed along the link route. You can specify a prefix to match an exact address or beginning segment of an address. Alternatively, you can specify a pattern to match an address using wildcards.
A *prefix* matches either an exact address or the beginning segment within an address that is delimited by either a **.** or / character. For example, the prefix **my_address** would match the address **my_address** as well as **my_address.1** and **my_address/1**. However, it would not match **my_address1**.

A *pattern* matches an address that corresponds to a pattern. A pattern is a sequence of words delimited by either a **.** or / character. You can use wildcard characters to represent a word. The * character matches exactly one word, and the **#** character matches any sequence of zero or more words.

The * and **#** characters are reserved as wildcards. Therefore, you should not use them in the message address.

For more information about creating address patterns, see Section 12.1.5, “Address pattern matching”.

NOTE

You can convert a **prefix** value to a **pattern** by appending **/#** to it. For example, the prefix **a/b/c** is equivalent to the pattern **a/b/c/#**.

connection | containerID

How the router should connect to the broker. You can specify either an outgoing connection (**connection**) or the container ID of the broker (**containerID**).
If multiple brokers are connected to the router through this connection, requests for addresses matching the link route’s prefix or pattern are balanced across the brokers. Alternatively, if you want to specify a particular broker, use **containerID** and add the broker’s container ID.

direction

Set this attribute to **in** to specify that clients can send messages into the router network on this link route.

For information about additional attributes, see linkRoute in the **qdrouterd.conf** man page.

4. If you want clients to receive messages on this link route, create an outgoing link route:

```
linkRoute {
    prefix: my_queue
    connection: my_broker
    direction: out
    ...
}
```

prefix | pattern

The address prefix or pattern that matches the broker queue from which you want to receive routed link-attaches. All messages that match this prefix or pattern will be distributed along the link route. You can specify a prefix to match an exact address or beginning segment of an address. Alternatively, you can specify a pattern to match an address using wildcards.
A *prefix* matches either an exact address or the beginning segment within an address that is delimited by either a **.** or / character. For example, the prefix **my_address** would match the address **my_address** as well as **my_address.1** and **my_address/1**. However, it would not match **my_address1**.

A *pattern* matches an address that corresponds to a pattern. A pattern is a sequence of words delimited by either a **.** or / character. You can use wildcard characters to represent a word. The ***** character matches exactly one word, and the **#** character matches any sequence of zero or more words.

The ***** and **#** characters are reserved as wildcards. Therefore, you should not use them in the message address.

For more information about creating address patterns, see Section 12.1.5, "Address pattern matching".

NOTE

You can convert a **prefix** value to a **pattern** by appending **/#** to it. For example, the prefix **a/b/c** is equivalent to the pattern **a/b/c/#**.

connection | containerID

How the router should connect to the broker. You can specify either an outgoing connection (**connection**) or the container ID of the broker (**containerID**).
If multiple brokers are connected to the router through this connection, requests for addresses matching the link route's prefix or pattern are balanced across the brokers. Alternatively, if you want to specify a particular broker, use **containerID** and add the broker's container ID.

direction

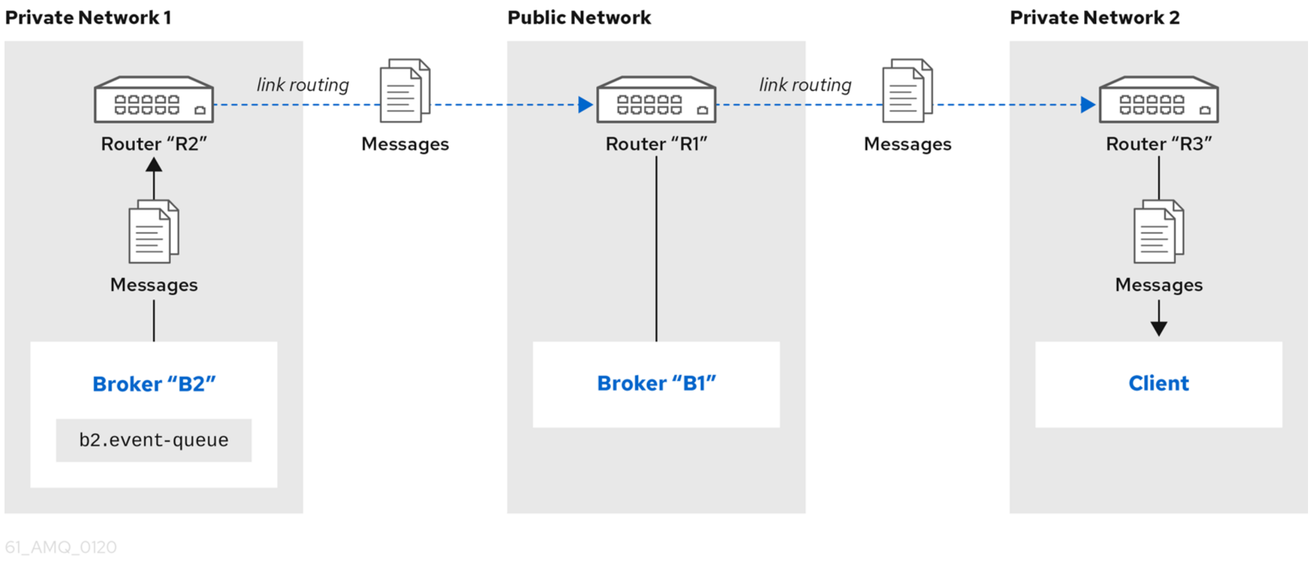
Set this attribute to **out** to specify that this link route is for receivers.

For information about additional attributes, see linkRoute in the **qdrouterd.conf** man page.

12.2.3. Link route example: Connecting clients and brokers on different networks

This example shows how a link route can connect a client to a message broker that is on a different private network.

Figure 12.6. Router network with isolated clients



The client is constrained by firewall policy to connect to the router in its own network (**R3**). However, it can use a link route to access queues, topics, and any other AMQP services that are provided on message brokers **B1** and **B2** – even though they are on different networks.

In this example, the client needs to receive messages from **b2.event-queue**, which is hosted on broker **B2** in **Private Network 1**. A link route connects the client and broker even though neither of them is aware that there is a router network between them.

Router configuration

To enable the client to receive messages from **b2.event-queue** on broker **B2**, router **R2** must be able to do the following:

- Connect to broker **B2**
- Route links to and from broker **B2**
- Advertise itself to the router network as a valid destination for links that have a **b2.event-queue** address

The relevant part of the configuration file for router **R2** shows the following:

```
connector { (1)
   name: broker
   role: route-container
   host: 192.0.2.1
   port: 61617
   saslMechanisms: ANONYMOUS
}

linkRoute { (2)
   prefix: b2
   direction: in
   connection: broker
}

linkRoute { (3)
   prefix: b2
```

```
    direction: out
    connection: broker
}
```

1 The outgoing connection from the router to broker **B2**. The **route-container** role enables the router to connect to an external AMQP container (in this case, a broker).

2 The incoming link route for receiving links from client senders. Any sender with a target whose address begins with **b2** will be routed to broker **B2** using the **broker** connector.

3 The outgoing link route for sending links to client receivers. Any receivers whose source address begins with **b2** will be routed to broker **B2** using the **broker** connector.

This configuration enables router **R2** to advertise itself as a valid destination for targets and sources starting with **b2**. It also enables the router to connect to broker **B2**, and to route links to and from queues starting with the **b2** prefix.

NOTE

While not required, routers **R1** and **R3** should also have the same configuration.

How the client receives messages
By using the configured link route, the client can receive messages from broker **B2** even though they are on different networks.

Router **R2** establishes a connection to broker **B2**. Once the connection is open, **R2** tells the other routers (**R1** and **R3**) that it is a valid destination for link routes to the **b2** prefix. This means that sender and receiver links attached to **R1** or **R3** will be routed along the shortest path to **R2**, which then routes them to broker **B2**.

To receive messages from the **b2.event-queue** on broker **B2**, the client attaches a receiver link with a source address of **b2.event-queue** to its local router, **R3**. Because the address matches the **b2** prefix, **R3** routes the link to **R1**, which is the next hop in the route to its destination. **R1** routes the link to **R2**, which routes it to broker **B2**. The client now has a receiver established, and it can begin receiving messages.

NOTE

If broker **B2** is unavailable for any reason, router **R2** will not advertise itself as a destination for **b2** addresses. In this case, routers **R1** and **R3** will reject link attaches that should be routed to broker **B2** with an error message indicating that there is no route available to the destination.

PART VI. MANAGE

CHAPTER 13. MONITORING USING AMQ MANAGEMENT CONSOLE

AMQ Management Console is a web console for monitoring the status and performance of AMQ Interconnect router networks.

Prerequisites

- AMQ Management Console requires the **qpid-dispatch-console** package.
  For more information about installing packages, see the Chapter 5, *Installing AMQ Interconnect* .

13.1. SETTING UP ACCESS TO AMQ MANAGEMENT CONSOLE

Before you can access the web console, you must configure a **listener** to accept HTTP connections for the web console and serve the console files.

Procedure

1. On the router from which you want to access the web console, open the **/etc/qpid-dispatch/qdrouterd.conf** configuration file.

2. Add a **listener** to serve the console.
   This example creates a **listener** that clients can use to access the web console:

   ```
   listener {
       host: 0.0.0.0
       port: 8672
       role: normal
       http: true
       httpRootDir: /usr/share/qpid-dispatch/console
   }
   ```

   host

   The IP address (IPv4 or IPv6) or hostname on which the router will listen.

   port

   The port number or symbolic service name on which the router will listen.

   role

   The role of the connection. Specify **normal** to indicate that this connection is used for client traffic.

   http

   Set this attribute to **true** to specify that this **listener** should accept HTTP connections instead of plain AMQP connections.

   httpRootDir

   Specify the absolute path to the directory that contains the web console HTML files. The default directory is the stand-alone console installation directory, usually **/usr/share/qpid-dispatch/console**.

3. If you want to secure access to the console, secure the **listener**.
   For more information, see Section 9.2, "Securing incoming client connections" . This example adds basic user name and password authentication using SASL PLAIN:

   –

```
listener {
    host: 0.0.0.0
    port: 8672
    role: normal
    http: true
    httpRootDir: /usr/share/qpid-dispatch/console
    authenticatePeer: yes
    saslMechanisms: PLAIN
}
```

4. If you want to set up access to the web console from any other router in the router network, repeat this procedure for each router.

13.2. ACCESSING AMQ MANAGEMENT CONSOLE

You can access the web console from a web browser.

Procedure

1. In a web browser, navigate to the web console URL.
   The web console URL is the *<host>*:*<port>* from the **listener** that you created to serve the web console. For example: **localhost:8672**.

   The AMQ Management Console opens. If you set up user name and password authentication, the **Connect** tab is displayed.

2. If necessary, log in to the web console.
   If you set up user name and password authentication, enter your user name and password to access the web console.

   The syntax for the user name is <*user*>@<*domain*>. For example: **admin@my-domain**.

13.3. MONITORING THE ROUTER NETWORK USING AMQ MANAGEMENT CONSOLE

The web console provides several sections that you can use to monitor the router network.

This section...	Provides...
Overview	Aggregated information about the router network. This information includes the following: • Dashboard (shows router network statistics) • Routers • Addresses • Links • Connections • Logs

This section...	Provides...
Visualizations	Graphical view of the router network. You can see the following types of visualizations: **Topology** Topology of the router network, including routers, clients, and brokers. This visualization also shows how messages are flowing through the network. **Message flow** A chord diagram showing the real-time message flow by address.
Details	Detailed configuration information about each AMQP management entity, for each router in the router network. You can view and change the configuration of any of the routers in the network.

CHAPTER 14. MONITORING USING QDSTAT

The **qdstat** tool is a command-line tool for monitoring the status and performance of AMQ Interconnect router networks.

14.1. SYNTAX FOR USING QDSTAT

You can use **qdstat** with the following syntax:

```
$ qdstat <option> [<connection-options>] [<secure-connection-options>]
```

This specifies:

- An *option* for the type of information to view.
- One or more optional *connection options* to specify a router for which to view the information. If you do not specify a connection option, **qdstat** connects to the router listening on localhost and the default AMQP port (5672).
- The *secure connection options* if the router for which you want to view information only accepts secure connections.

Additional resources

- For more information about **qdstat**, see the qdstat man page.

14.2. COMMANDS FOR MONITORING THE ROUTER NETWORK

You can use **qdstat** to view the status of routers on your router network. For example, you can view information about the attached links and configured addresses, available connections, and nodes in the router network.

To...	Use this command...
Create a state dump containing all statistics for all routers A state dump shows the current operational state of the router network.	`$ qdstat --all-routers --all-entities` If you run this command on an interior router, it displays the statistics for all interior routers. If you run the command on an edge router, it displays the statistics for only that edge router.
Create a state dump containing a single statistic for all routers	`$ qdstat -l|-a|-c|--autolinks|--linkroutes|-g|-m --all-routers` If you run this command on an interior router, it displays the statistic for all interior routers. If you run the command on an edge router, it displays the statistic for only that edge router.

To...	Use this command...
Create a state dump containing all statistics for a single router	`$ qdstat --all-entities` This command shows the statistics for the local router only.
View general statistics for a router	`$ qdstat -g [all-routers|<connection-options>]`
View a list of connections to a router	`$ qdstat -c [all-routers|<connection-options>]`
View the AMQP links attached to a router You can view a list of AMQP links attached to the router from clients (sender/receiver), from or to other routers into the network, to other containers (for example, brokers), and from the tool itself.	`$ qdstat -l [all-routers|<connection-options>]`
View known routers on the router network	`$ qdstat -n [all-routers|<connection-options>]`
View the addresses known to a router	`$ qdstat -a [all-routers|<connection-options>]`
View a router’s autolinks	`$ qdstat --autolinks [all-routers|<connection-options>]`
View the status of a router’s link routes	`$ qdstat --linkroutes [all-routers|<connection-options>]`
View a router’s policy global settings and statistics	`$ qdstat --policy [all-routers|<connection-options>]`
View a router’s policy vhost settings	`$ qdstat --vhosts [all-routers|<connection-options>]`
View a router’s policy vhost statistics	`$ qdstat --vhoststats [all-routers|<connection-options>]`

To…	Use this command…
View a router’s vhostgroup settings	`$ qdstat --vhostgroups [all-routers|<connection-options>]`
View a router’s memory consumption	`$ qdstat -m [all-routers|<connection-options>]`

Additional resources

- For more information about the fields displayed by each **qdstat** command, see the qdstat man page.

CHAPTER 15. MANAGING USING QDMANAGE

The **qdmanage** tool is a command-line tool for viewing and modifying the configuration of a running router at runtime.

NOTE

If you make a change to a router using **qdmanage**, the change takes effect immediately, but is lost if the router is stopped. If you want to make a permanent change to a router's configuration, you must edit the router's **/etc/qpid-dispatch/qdrouterd.conf** configuration file.

You can use **qdmanage** with the following syntax:

```
$ qdmanage [<connection-options>] <operation> [<options>]
```

This specifies:

- One or more optional *connection options* to specify the router on which to perform the operation, or to supply security credentials if the router only accepts secure connections.
  If you do not specify any connection options, **qdmanage** connects to the router listening on localhost and the default AMQP port (5672).
- The *operation* to perform on the router.
- One or more optional *options* to specify a configuration entity on which to perform the operation or how to format the command output.

When you enter a **qdmanage** command, it is executed as an AMQP management operation request, and then the response is returned as command output in JSON format.

For example, the following command executes a query operation on a router, and then returns the response in JSON format:

```
$ qdmanage query --type listener
[
  {
    "stripAnnotations": "both",
    "addr": "127.0.0.1",
    "multiTenant": false,
    "requireSsl": false,
    "idleTimeoutSeconds": 16,
    "saslMechanisms": "ANONYMOUS",
    "maxFrameSize": 16384,
    "requireEncryption": false,
    "host": "0.0.0.0",
    "cost": 1,
    "role": "normal",
    "http": false,
    "maxSessions": 32768,
    "authenticatePeer": false,
    "type": "org.apache.qpid.dispatch.listener",
    "port": "amqp",
    "identity": "listener/0.0.0.0:amqp",
    "name": "listener/0.0.0.0:amqp"
  }
]
```

Additional resources

- For more information about **qdmanage**, see the qdmanage man page.

CHAPTER 16. TROUBLESHOOTING AMQ INTERCONNECT

You can use the AMQ Interconnect logs to diagnose and troubleshoot error and performance issues with the routers in your router network.

16.1. VIEWING LOG ENTRIES

You may need to view log entries to diagnose errors, performance problems, and other important issues. A log entry consists of an optional timestamp, the logging module, the logging level, and the log message.

Procedure

- Do one of the following:
  - View log entries on the console.
    By default, events are logged to the console, and you can view them there. However, if the **output** attribute is set for a particular logging module, then you can find those log entries in the specified location (**stderr**, **syslog**, or a file).
  - Use the **qdstat --log** command to view recent log entries.
    You can use the **--limit** parameter to limit the number of log entries that are displayed. For more information about **qdstat**, see qdstat man page.

    This example displays the last three log entries for **Router.A**:

    ```
    $ qdstat --log --limit=3 -r ROUTER.A
    Wed Jun  7 17:49:32 2019 ROUTER (none) Core action 'link_deliver'
    Wed Jun  7 17:49:32 2019 ROUTER (none) Core action 'send_to'
    Wed Jun  7 17:49:32 2019 SERVER (none) [2]:0 -> @flow(19) [next-incoming-id=1,
    incoming-window=61, next-outgoing-id=0, outgoing-window=2147483647, handle=0,
    delivery-count=1, link-credit=250, drain=false]
    ```

NOTE

vhost entries are only populated if **multiTenant** is set to **true** in the **/etc/qpid-dispatch/qdrouterd.conf** configuration file.

Additional resources

- For more information about configuring logging modules, see Section 11.2, "Configuring default logging".

16.2. TROUBLESHOOTING USING LOGS

You can use AMQ Interconnect log entries to help diagnose error and performance issues with the routers in your network.

Example 16.1. Troubleshooting connections and links

In this example, **ROUTER** logs show the lifecycle of a connection and a link that is associated with it.

```
2019-04-05 14:54:38.037248 -0400 ROUTER (info) [C1] Connection Opened: dir=in
```

```
host=127.0.0.1:55440 vhost= encrypted=no auth=no user=anonymous container_id=95e55424-
6c0a-4a5c-8848-65a3ea5cc25a props= (1)
2019-04-05 14:54:38.038137 -0400 ROUTER (info) [C1][L6] Link attached: dir=in source={<none>
expire:sess} target={$management expire:sess} (2)
2019-04-05 14:54:38.041103 -0400 ROUTER (info) [C1][L6] Link lost: del=1 presett=0 psdrop=0
acc=1 rej=0 rel=0 mod=0 delay1=0 delay10=0 (3)
2019-04-05 14:54:38.041154 -0400 ROUTER (info) [C1] Connection Closed (4)
```

1. The connection is opened. Each connection has a unique ID (**C1**). The log also shows some information about the connection.
2. A link is attached over the connection. The link is identified with a unique ID (**L6**). The log also shows the direction of the link, and the source and target addresses.
3. The link is detached. The log shows the link's terminal statistics.
4. The connection is closed.

NOTE

If necessary, you can use **qdmanage** to enable protocol-level trace logging for a particular connection. You can use this to trace the AMQP frames. For example:

```
$ qdmanage update --type=connection --id=C1 enableProtocolTrace=true
```

Example 16.2. Troubleshooting the network topology

In this example, on **Router.A**, the **ROUTER_HELLO** logs show that it is connected to **Router.B**, and that **Router.B** is connected to **Router.A** and **Router.C**:

```
Tue Jun  7 13:50:21 2016 ROUTER_HELLO (trace) RCVD: HELLO(id=Router.B area=0
inst=1465307413 seen=['Router.A', 'Router.C']) (1)
Tue Jun  7 13:50:21 2016 ROUTER_HELLO (trace) SENT: HELLO(id=Router.A area=0
inst=1465307416 seen=['Router.B']) (2)
Tue Jun  7 13:50:22 2016 ROUTER_HELLO (trace) RCVD: HELLO(id=Router.B area=0
inst=1465307413 seen=['Router.A', 'Router.C'])
Tue Jun  7 13:50:22 2016 ROUTER_HELLO (trace) SENT: HELLO(id=Router.A area=0
inst=1465307416 seen=['Router.B'])
```

1. **Router.A** received a Hello message from **Router.B**, which can see **Router.A** and **Router.C**.
2. **Router.A** sent a Hello message to **Router.B**, which is the only router it can see.

On **Router.B**, the **ROUTER_HELLO** log shows the same router topology from a different perspective:

```
Tue Jun  7 13:50:18 2016 ROUTER_HELLO (trace) SENT: HELLO(id=Router.B area=0
inst=1465307413 seen=['Router.A', 'Router.C']) (1)
Tue Jun  7 13:50:18 2016 ROUTER_HELLO (trace) RCVD: HELLO(id=Router.A area=0
inst=1465307416 seen=['Router.B']) 2
Tue Jun  7 13:50:19 2016 ROUTER_HELLO (trace) RCVD: HELLO(id=Router.C area=0
inst=1465307411 seen=['Router.B']) 3
```

1. **Router.B** sent a Hello message to **Router.A** and **Router.C**.
2. **Router.B** received a Hello message from **Router.A**, which can only see **Router.B**.
3. **Router.B** received a Hello message from **Router.C**, which can only see **Router.B**.

Example 16.3. Tracing the link state between routers

Periodically, each router sends a Link State Request (LSR) to the other routers and receives a Link State Update (LSU) with the requested information. Exchanging the above information, each router can compute the next hops in the topology, and the related costs.

In this example, the **ROUTER_LS** logs show the RA, LSR, and LSU messages sent between three routers:

```
Tue Jun  7 14:10:02 2016 ROUTER_LS (trace) SENT: LSR(id=Router.A area=0) to: Router.C
Tue Jun  7 14:10:02 2016 ROUTER_LS (trace) SENT: LSR(id=Router.A area=0) to: Router.B
Tue Jun  7 14:10:02 2016 ROUTER_LS (trace) SENT: RA(id=Router.A area=0 inst=1465308600
ls_seq=1 mobile_seq=1) 1
Tue Jun  7 14:10:02 2016 ROUTER_LS (trace) RCVD: LSU(id=Router.B area=0
inst=1465308595 ls_seq=2 ls=LS(id=Router.B area=0 ls_seq=2 peers={'Router.A': 1L, 'Router.C':
1L})) 2
Tue Jun  7 14:10:02 2016 ROUTER_LS (trace) RCVD: LSR(id=Router.B area=0)
Tue Jun  7 14:10:02 2016 ROUTER_LS (trace) SENT: LSU(id=Router.A area=0 inst=1465308600
ls_seq=1 ls=LS(id=Router.A area=0 ls_seq=1 peers={'Router.B': 1}))
Tue Jun  7 14:10:02 2016 ROUTER_LS (trace) RCVD: RA(id=Router.C area=0 inst=1465308592
ls_seq=1 mobile_seq=0)
Tue Jun  7 14:10:02 2016 ROUTER_LS (trace) SENT: LSR(id=Router.A area=0) to: Router.C
Tue Jun  7 14:10:02 2016 ROUTER_LS (trace) RCVD: LSR(id=Router.C area=0) 3
Tue Jun  7 14:10:02 2016 ROUTER_LS (trace) SENT: LSU(id=Router.A area=0 inst=1465308600
ls_seq=1 ls=LS(id=Router.A area=0 ls_seq=1 peers={'Router.B': 1}))
Tue Jun  7 14:10:02 2016 ROUTER_LS (trace) RCVD: LSU(id=Router.C area=0
inst=1465308592 ls_seq=1 ls=LS(id=Router.C area=0 ls_seq=1 peers={'Router.B': 1L})) 4
Tue Jun  7 14:10:03 2016 ROUTER_LS (trace) Computed next hops: {'Router.C': 'Router.B',
'Router.B': 'Router.B'} 5
Tue Jun  7 14:10:03 2016 ROUTER_LS (trace) Computed costs: {'Router.C': 2L, 'Router.B': 1}
Tue Jun  7 14:10:03 2016 ROUTER_LS (trace) Computed valid origins: {'Router.C': [], 'Router.B':
[]}
```

1. **Router.A** sent LSR requests and an RA advertisement to the other routers on the network.
2. **Router.A** received an LSU from **Router.B**, which has two peers: **Router.A**, and **Router.C** (with a cost of **1**).
3. **Router.A** received an LSR from both **Router.B** and **Router.C**, and replied with an LSU.
4. **Router.A** received an LSU from **Router.C**, which only has one peer: **Router.B** (with a cost of **1**).
5. After the LSR and LSU messages are exchanged, **Router.A** computed the router topology with

the related costs.

Example 16.4. Tracing the state of mobile addresses attached to a router

In this example, the **ROUTER_MA** logs show the Mobile Address Request (MAR) and Mobile Address Update (MAU) messages sent between three routers:

```
Tue Jun  7 14:27:20 2016 ROUTER_MA (trace) SENT: MAU(id=Router.A area=0 mobile_seq=1
add=['Cmy_queue', 'Dmy_queue', 'M0my_queue_wp'] del=[]) 1
Tue Jun  7 14:27:21 2016 ROUTER_MA (trace) RCVD: MAR(id=Router.C area=0 have_seq=0)
2
Tue Jun  7 14:27:21 2016 ROUTER_MA (trace) SENT: MAU(id=Router.A area=0 mobile_seq=1
add=['Cmy_queue', 'Dmy_queue', 'M0my_queue_wp'] del=[])
Tue Jun  7 14:27:22 2016 ROUTER_MA (trace) RCVD: MAR(id=Router.B area=0 have_seq=0)
3
Tue Jun  7 14:27:22 2016 ROUTER_MA (trace) SENT: MAU(id=Router.A area=0 mobile_seq=1
add=['Cmy_queue', 'Dmy_queue', 'M0my_queue_wp'] del=[])
Tue Jun  7 14:27:39 2016 ROUTER_MA (trace) RCVD: MAU(id=Router.C area=0 mobile_seq=1
add=['M0my_test'] del=[]) 4
Tue Jun  7 14:27:51 2016 ROUTER_MA (trace) RCVD: MAU(id=Router.C area=0 mobile_seq=2
add=[] del=['M0my_test']) 5
```

1 **Router.A** sent MAU messages to the other routers in the network to notify them about the addresses added for **my_queue** and **my_queue_wp**.

2 **Router.A** received a MAR message in response from **Router.C**.

3 **Router.A** received another MAR message in response from **Router.B**.

4 **Router.C** sent a MAU message to notify the other routers that it added and address for **my_test**.

5 **Router.C** sent another MAU message to notify the other routers that it deleted the address for **my_test** (because the receiver is detached).

Example 16.5. Finding information about messages sent and received by a router

In this example, the **MESSAGE** logs show that **Router.A** has sent and received some messages related to the Hello protocol, and sent and received some other messages on a link for a mobile address:

```
Tue Jun  7 14:36:54 2016 MESSAGE (trace) Sending
Message{to='amqp:/_topo/0/Router.B/qdrouter'
body='\d1\00\00\00\1b\00\00\00\04\a1\02id\a1\08R'} on link qdlink.p9XmBm19uDqx50R
Tue Jun  7 14:36:54 2016 MESSAGE (trace) Received
Message{to='amqp:/_topo/0/Router.A/qdrouter' body='\d1\00\00\00\8e\00\00\00
\a1\06ls_se'} on link qdlink.phMsJOq7YaFsGAG
Tue Jun  7 14:36:54 2016 MESSAGE (trace) Received Message{
body='\d1\00\00\00\10\00\00\00\02\a1\08seque'} on link qdlink.FYHqBX+TtwXZHfV
Tue Jun  7 14:36:54 2016 MESSAGE (trace) Sending Message{
body='\d1\00\00\00\10\00\00\00\02\a1\08seque'} on link qdlink.yU1tnPs5KbMlieM
```

```
Tue Jun  7 14:36:54 2016 MESSAGE (trace) Sending Message{to='amqp:/_local/qdhello'
body='\d1\00\00\00G\00\00\00\08\a1\04seen\d0'} on link qdlink.p9XmBm19uDqx50R
Tue Jun  7 14:36:54 2016 MESSAGE (trace) Sending
Message{to='amqp:/_topo/0/Router.C/qdrouter'
body='\d1\00\00\00\1b\00\00\00\04\a1\02id\a1\08R'} on link qdlink.p9XmBm19uDqx50R
```

Example 16.6. Tracking configuration changes to a router

In this example, the **AGENT** logs show that on **Router.A**, **address**, **linkRoute**, and **autoLink** entities were added to the router's configuration file. When the router was started, the **AGENT** module applied these changes, and they are now viewable in the log:

```
Tue Jun  7 15:07:32 2016 AGENT (debug) Add entity: ConnectorEntity(addr=127.0.0.1,
allowRedirect=True, cost=1, host=127.0.0.1, identity=connector/127.0.0.1:5672:BROKER,
idleTimeoutSeconds=16, maxFrameSize=65536, name=BROKER, port=5672, role=route-
container, stripAnnotations=both, type=org.apache.qpid.dispatch.connector,
verifyHostname=True)
Tue Jun  7 15:07:32 2016 AGENT (debug) Add entity:
RouterConfigAddressEntity(distribution=closest, identity=router.config.address/0,
name=router.config.address/0, prefix=my_address,
type=org.apache.qpid.dispatch.router.config.address, waypoint=False)
Tue Jun  7 15:07:32 2016 AGENT (debug) Add entity:
RouterConfigAddressEntity(distribution=balanced, identity=router.config.address/1,
name=router.config.address/1, prefix=my_queue_wp,
type=org.apache.qpid.dispatch.router.config.address, waypoint=True)
Tue Jun  7 15:07:32 2016 AGENT (debug) Add entity:
RouterConfigLinkrouteEntity(connection=BROKER, direction=in, distribution=linkBalanced,
identity=router.config.linkRoute/0, name=router.config.linkRoute/0, prefix=my_queue,
type=org.apache.qpid.dispatch.router.config.linkRoute)
Tue Jun  7 15:07:32 2016 AGENT (debug) Add entity:
RouterConfigLinkrouteEntity(connection=BROKER, direction=out, distribution=linkBalanced,
identity=router.config.linkRoute/1, name=router.config.linkRoute/1, prefix=my_queue,
type=org.apache.qpid.dispatch.router.config.linkRoute)
Tue Jun  7 15:07:32 2016 AGENT (debug) Add entity:
RouterConfigAutolinkEntity(address=my_queue_wp, connection=BROKER, direction=in,
identity=router.config.autoLink/0, name=router.config.autoLink/0,
type=org.apache.qpid.dispatch.router.config.autoLink)
Tue Jun  7 15:07:32 2016 AGENT (debug) Add entity:
RouterConfigAutolinkEntity(address=my_queue_wp, connection=BROKER, direction=out,
identity=router.config.autoLink/1, name=router.config.autoLink/1,
type=org.apache.qpid.dispatch.router.config.autoLink)
```

Example 16.7. Troubleshooting policy and vhost access rules

In this example, the **POLICY** logs show that this router has no limits on maximum connections, and the default application policy is disabled:

```
Tue Jun  7 15:07:32 2016 POLICY (info) Policy configured maximumConnections: 0, policyFolder:
'', access rules enabled: 'false'
Tue Jun  7 15:07:32 2016 POLICY (info) Policy fallback defaultApplication is disabled
```

Example 16.8. Diagnosing errors

In this example, the **ERROR** logs show that the router failed to start when an incorrect path was specified for the router's configuration file:

```
$ qdrouterd --conf my_config
Wed Jun 15 09:53:28 2016 ERROR (error) Python: Exception: Cannot load configuration file
my_config: [Errno 2] No such file or directory: 'my_config'
Wed Jun 15 09:53:28 2016 ERROR (error) Traceback (most recent call last):
  File "/usr/lib/qpid-dispatch/python/qpid_dispatch_internal/management/config.py", line 155, in
configure_dispatch
    config = Config(filename)
  File "/usr/lib/qpid-dispatch/python/qpid_dispatch_internal/management/config.py", line 41, in
__init__
    self.load(filename, raw_json)
  File "/usr/lib/qpid-dispatch/python/qpid_dispatch_internal/management/config.py", line 123, in
load
    with open(source) as f:
Exception: Cannot load configuration file my_config: [Errno 2] No such file or directory: 'my_config'

Wed Jun 15 09:53:28 2016 MAIN (critical) Router start-up failed: Python: Exception: Cannot load
configuration file my_config: [Errno 2] No such file or directory: 'my_config'
qdrouterd: Python: Exception: Cannot load configuration file my_config: [Errno 2] No such file or
directory: 'my_config'
```

Additional resources

- For more information about logging modules, see Section 11.1, "Logging modules".

APPENDIX A. USING YOUR SUBSCRIPTION

AMQ is provided through a software subscription. To manage your subscriptions, access your account at the Red Hat Customer Portal.

A.1. ACCESSING YOUR ACCOUNT

Procedure

1. Go to access.redhat.com.
2. If you do not already have an account, create one.
3. Log in to your account.

A.2. ACTIVATING A SUBSCRIPTION

Procedure

1. Go to access.redhat.com.
2. Navigate to **My Subscriptions**.
3. Navigate to **Activate a subscription** and enter your 16-digit activation number.

A.3. DOWNLOADING RELEASE FILES

To access .zip, .tar.gz, and other release files, use the customer portal to find the relevant files for download. If you are using RPM packages or the Red Hat Maven repository, this step is not required.

Procedure

1. Open a browser and log in to the Red Hat Customer Portal **Product Downloads** page at access.redhat.com/downloads.
2. Locate the **Red Hat AMQ** entries in the **INTEGRATION AND AUTOMATION** category.
3. Select the desired AMQ product. The **Software Downloads** page opens.
4. Click the **Download** link for your component.

A.4. REGISTERING YOUR SYSTEM FOR PACKAGES

To install RPM packages for this product on Red Hat Enterprise Linux, your system must be registered. If you are using downloaded release files, this step is not required.

Procedure

1. Go to access.redhat.com.
2. Navigate to **Registration Assistant**.
3. Select your OS version and continue to the next page.

4. Use the listed command in your system terminal to complete the registration.

For more information about registering your system, see one of the following resources:

- Red Hat Enterprise Linux 6 - Registering the system and managing subscriptions
- Red Hat Enterprise Linux 7 - Registering the system and managing subscriptions
- Red Hat Enterprise Linux 8 - Registering the system and managing subscriptions

Revised on 2020-10-08 11:31:52 UTC