Red Hat AMQ 7.2

Introducing Red Hat AMQ 7

Overview of Features and Components

Last Updated: 2018-07-16

Red Hat AMQ 7.2 Introducing Red Hat AMQ 7

Overview of Features and Components

Legal Notice

Abstract

This document highlights features and components of Red Hat AMQ 7. It also demonstrates common use cases and design patterns addressed in this product release.

Table of Contents

CHAPTER 1. ABOUT RED HAT AMQ 7

Red Hat AMQ provides fast, lightweight, and secure messaging for Internet-scale applications. AMQ Broker supports multiple protocols and fast message persistence. AMQ Interconnect leverages the AMQP protocol to distribute and scale your messaging resources across the network. AMQ Clients provides a suite of messaging APIs for multiple languages and platforms.

Think of the AMQ components as tools inside a toolbox. They can be used together or separately to build and maintain your messaging application, and AMQP is the glue in the toolbox that binds them together. AMQ components share a common management console, so you can manage them from a single interface.

1.1. KEY FEATURES

AMQ enables developers to build messaging applications that are fast, reliable, and easy to administer.

Messaging at Internet Scale
AMQ contains the tools to build advanced, multi-datacenter messaging networks. It can connect clients, brokers, and stand-alone services in a seamless messaging fabric.

Top-Tier Security and Performance
AMQ offers modern SSL/TLS encryption and extensible SASL authentication. AMQ delivers fast, high-volume messaging and class-leading JMS performance.

Broad Platform and Language Support
AMQ works with multiple languages and operating systems, so your diverse application components can communicate. AMQ supports C++, Java, JavaScript, Python, Ruby, and .NET applications, as well as Linux, Windows, and JVM-based environments.

Focused on Standards
AMQ implements the Java JMS 1.1 and 2.0 API specifications. Its components support the ISO-standard AMQP 1.0 and MQTT messaging protocols, as well as STOMP and WebSocket.

Centralized Management
With AMQ, you can administer all AMQ components from a single management interface. You can use JMX or the REST interface to manage servers programatically.

CHAPTER 2. COMPONENT OVERVIEW

Red Hat AMQ consists of AMQ Broker, AMQ Clients, and a new kind of messaging server called AMQ Interconnect. They work together to enable remote communication among distributed client applications. AMQ Console is the common interface used to monitor and manage AMQ deployments.

- AMQ Broker
- AMQ Clients
- AMQ Interconnect
- AMQ Console

2.1. AMQ BROKER

AMQ Broker is a full-featured, message-oriented middleware broker. It offers advanced addressing and queueing, fast message persistence, and high availability. AMQ Broker supports multiple protocols and operating environments, enabling you to use your existing assets. AMQ Broker supports integration with Red Hat JBoss Enterprise Application Platform.

See Using AMQ Broker for more information.

2.2. AMQ INTERCONNECT

AMQ Interconnect provides flexible routing of messages between AMQP-enabled endpoints, including clients, brokers, and standalone services. With a single connection into a network of AMQ Interconnect routers, a client can exchange messages with any other endpoint connected to the network.

AMQ Interconnect does not use master-slave clusters for high availability. It is typically deployed in topologies of multiple routers with redundant network paths, which it uses to provide reliable connectivity. AMQ Interconnect can distribute messaging workloads across the network and achieve new levels of scale with very low latency.

See Using AMQ Interconnect for more information.

2.3. AMQ CLIENTS

AMQ Clients is a collection of AMQP 1.0 messaging APIs for multiple languages and platforms. It includes JMS 2.0 support and new, event-driven APIs to enable integration into existing applications.

- Using the AMQ JMS Client
- Using the AMQ C++ Client
- Using the AMQ JavaScript Client
- Using the AMQ .NET Client
- Using the AMQ Python Client
- Using the AMQ Ruby Client

2.4. AMQ CONSOLE

AMQ Console is a tool for administering AMQ brokers and routers in one convenient interface. From AMQ Console you can manage addresses, queues, connections, and other administrative objects.

See Using AMQ Console for more information.

2.5. COMPONENT COMPATIBILITY

The following table lists the supported languages, platforms, and protocols of AMQ components. Note that any components supporting the same protocol can interoperate, even if their languages and platforms differ. For instance, AMQ Python can communicate with AMQ JMS.

Table 2.1. AMQ Component Compatibility

Component	Languages	Platforms	Protocols
AMQ Broker	-	JVM	AMQP 1.0, MQTT, OpenWire, STOMP, Core Protocol
AMQ Interconnect	-	Linux	AMQP 1.0
AMQ C++	C++	Linux, Windows	AMQP 1.0
AMQ JavaScript	JavaScript	Node.js, browsers	AMQP 1.0
AMQ JMS	Java	JVM	AMQP 1.0
AMQ .NET	C#	.NET	AMQP 1.0
AMQ Python	Python	Linux	AMQP 1.0
AMQ Ruby	Ruby	Linux	AMQP 1.0
AMQ Core Protocol JMS	Java	JVM	Core Protocol
AMQ OpenWire JMS	Java	JVM	OpenWire

See Red Hat AMQ 7 Supported Configurations for more information.

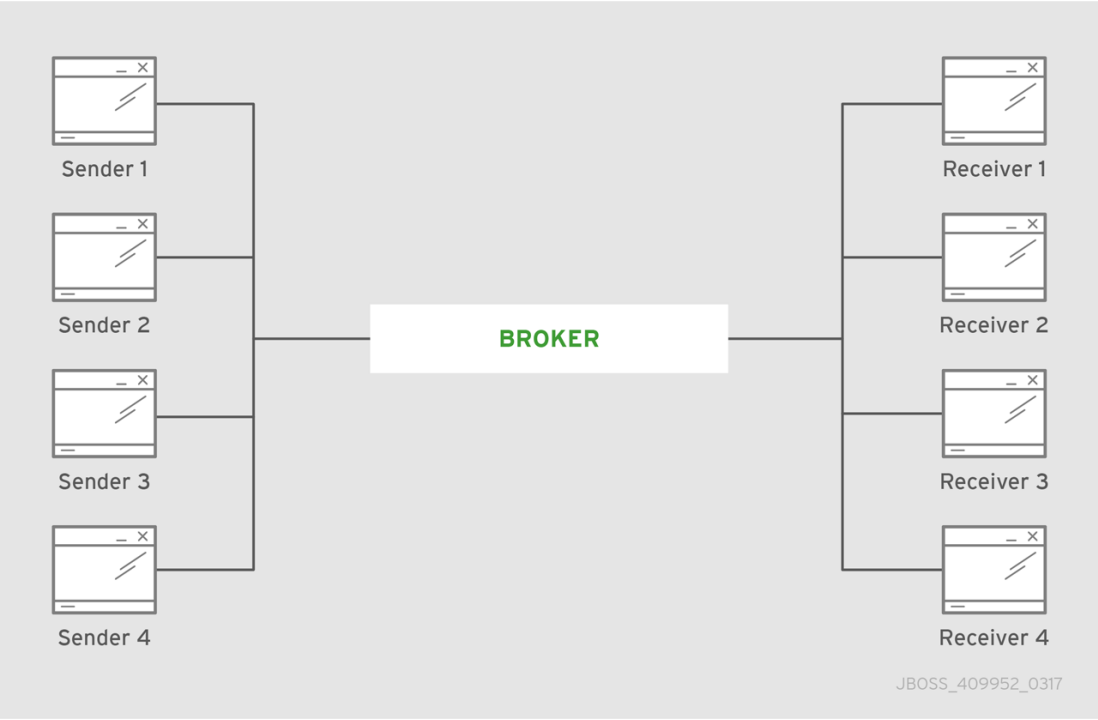
CHAPTER 3. COMMON DEPLOYMENT PATTERNS

Red Hat AMQ 7 can be set up in a large variety of topologies. The following are some of the common deployment patterns you can implement using AMQ components.

3.1. CENTRAL BROKER

The central broker pattern is relatively easy to set up and maintain. It is also relatively robust. Routes are typically local, because the broker and its clients are always within one network hop of each other, no matter how many nodes are added. This pattern is also known as *hub and spoke*, with the central broker as the hub and the clients the spokes.

Figure 3.1. Central Broker Pattern

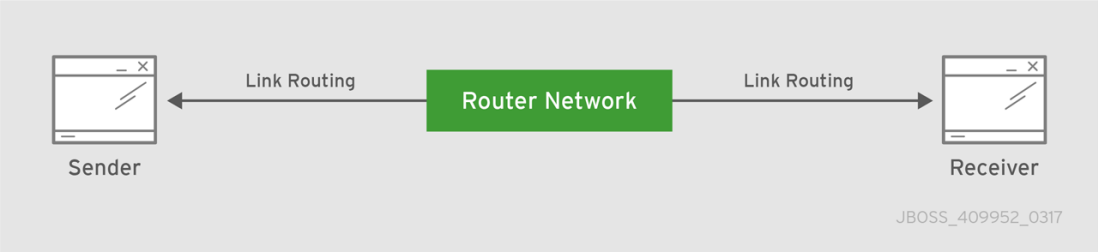


The only critical element is the central broker node. The focus of your maintenance efforts is on keeping this broker available to its clients.

3.2. ROUTED MESSAGING

When routing messages to remote destinations, the broker stores them in a local queue before forwarding them to their destination. However, sometimes an application requires sending request and response messages in real time, and having the broker store and forward messages is too costly. With AMQ you can use a router in place of a broker to avoid such costs. Unlike a broker, a router does not store messages before forwarding them to a destination. Instead, it works as a lightweight conduit and directly connects two endpoints.

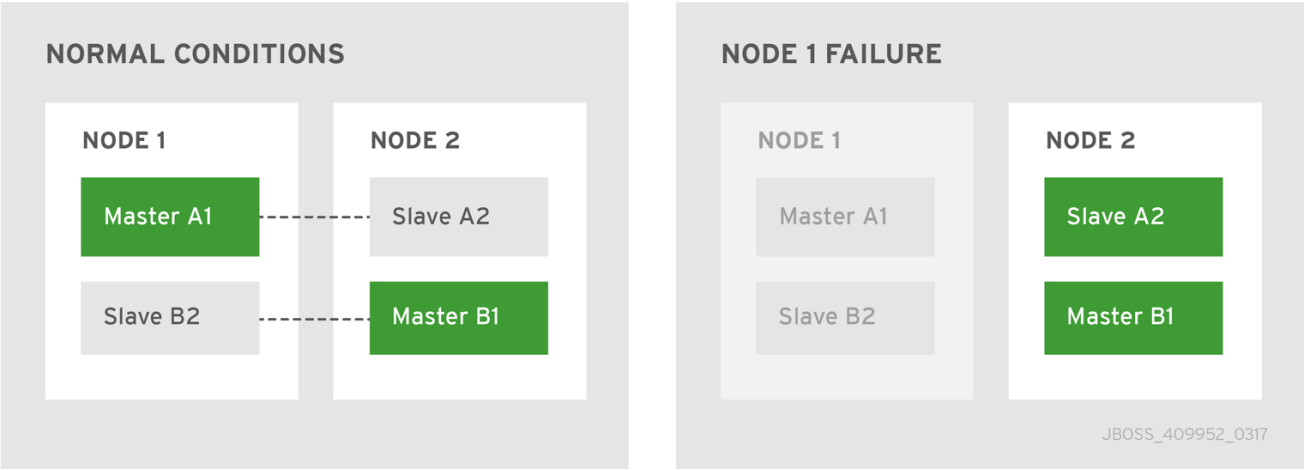
Figure 3.2. Brokerless Routed Messaging Pattern



3.3. HIGHLY AVAILABLE BROKERS

To ensure brokers are available for their clients, deploy a highly available (HA) master-slave pair to create a backup group. You might, for example, deploy two master-slave groups on two nodes. Such a deployment would provide a backup for each active broker, as seen in the following diagram.

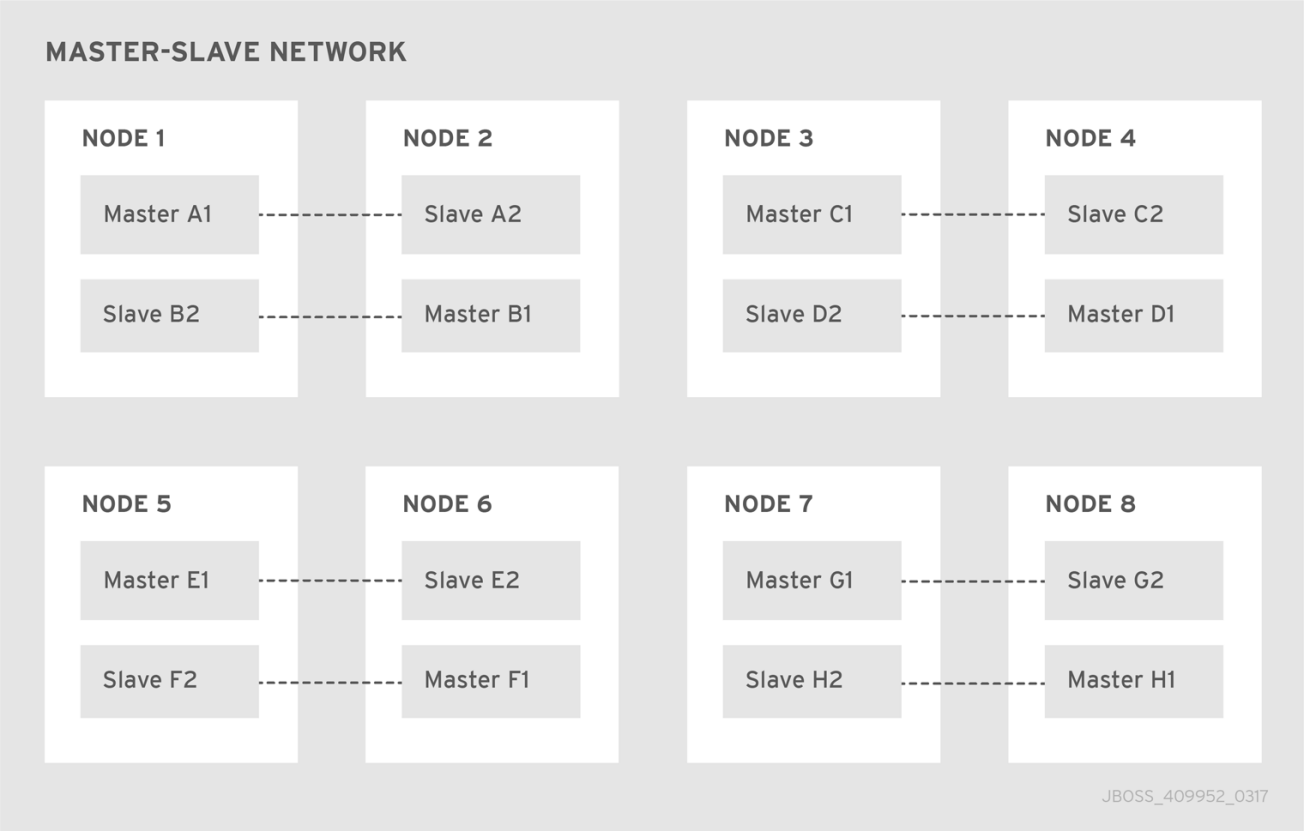
Figure 3.3. Master-Slave Pair



Under normal operating conditions one master broker is active on each node, which can be either a physical server or a virtual machine. If one node fails, the slave on the other node takes over. The result is two active brokers residing on the same healthy node.

By deploying master-slave pairs, you can scale out an entire network of such backup groups. Larger deployments of this type are useful for distributing the message processing load across many brokers. The broker network in the following diagram consists of eight master-slave groups distributed over eight nodes.

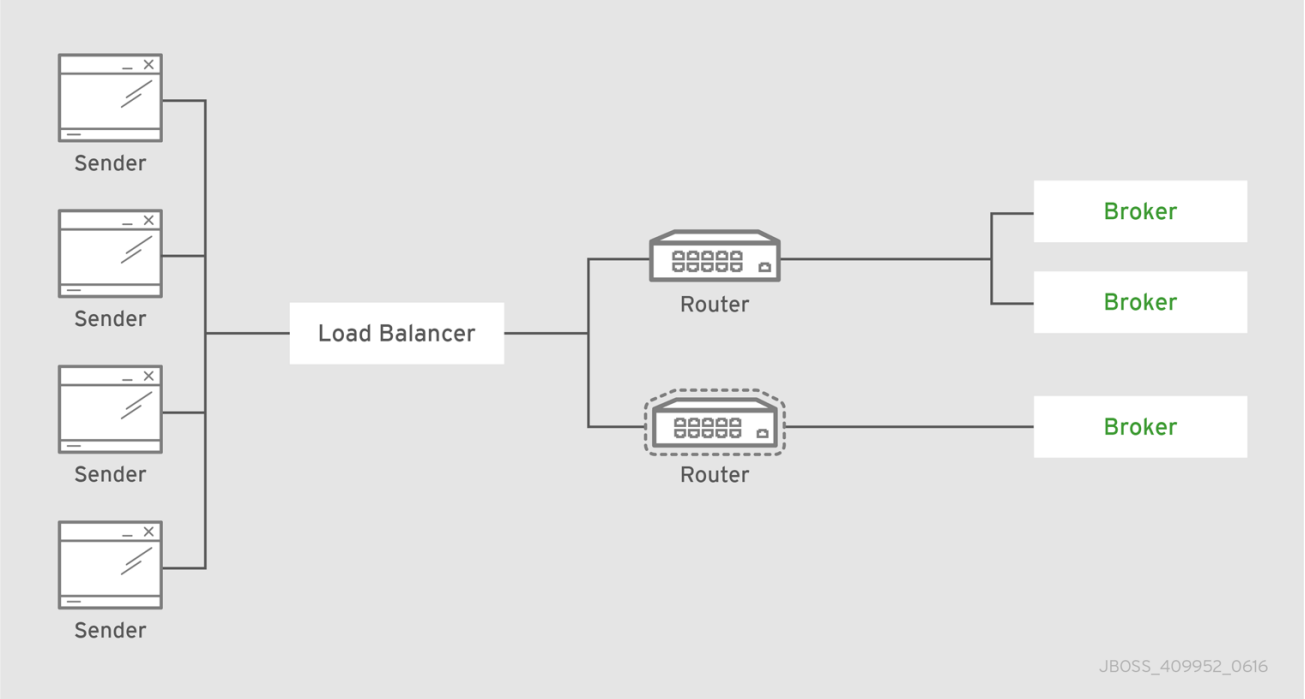
Figure 3.4. Master-Slave Network



3.4. ROUTER PAIR BEHIND A LOAD BALANCER

Deploying two routers behind a load balancer provides high availability, resiliency, and increased scalability for a single-datacenter deployment. Endpoints make their connections to a known URL, supported by the load balancer. Next, the load balancer spreads the incoming connections among the routers so that the connection and messaging load is distributed. If one of the routers fails, the endpoints connected to it will reconnect to the remaining active router.

Figure 3.5. Router Pair behind a Load Balancer

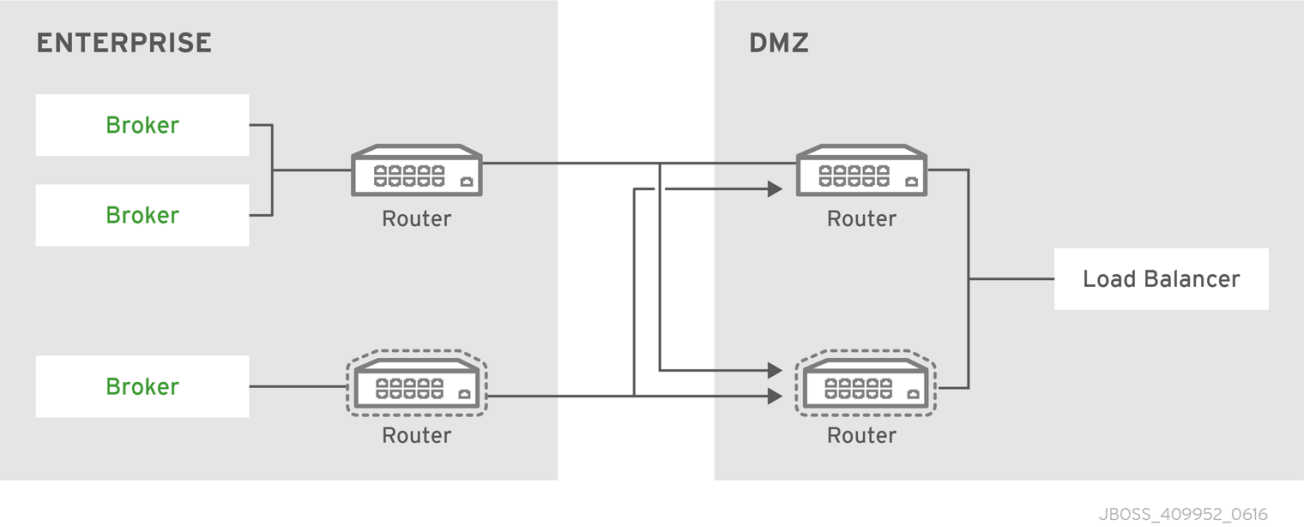


For even greater scalability, you can use a larger number of routers, three or four for example. Each router connects directly to all of the others.

3.5. ROUTER PAIR IN A DMZ

In this deployment architecture, the router network is providing a layer of protection and isolation between the clients in the outside world and the brokers backing an enterprise application.

Figure 3.6. Router Pair in a DMZ



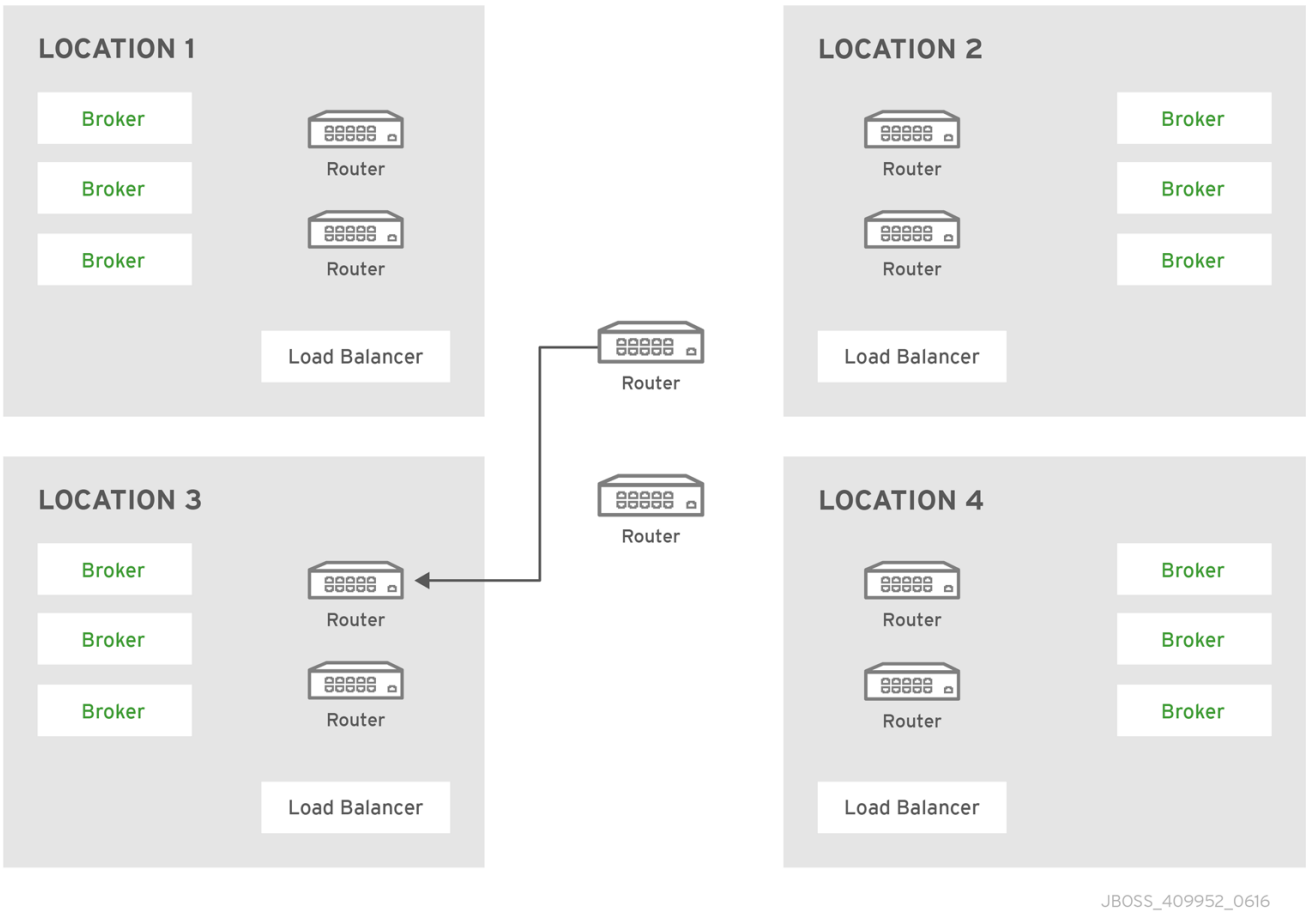
Important notes about the DMZ topology:

- Security for the connections within the deployment is separate from the security used for external clients. For example, your deployment might use a private Certificate Authority (CA) for internal security, issuing x.509 certificates to each router and broker for authentication, although external users might use a different, public CA.
- Inter-router connections between the enterprise and the DMZ are always established from the enterprise to the DMZ for security. Therefore, no connections are permitted from the outside into the enterprise. The AMQP protocol enables bi-directional communication after a connection is established, however.

3.6. ROUTER PAIRS IN DIFFERENT DATA CENTERS

You can use a more complex topology in a deployment of AMQ components that spans multiple locations. You can, for example, deploy a pair of load-balanced routers in each of four locations. You might include two backbone routers in the center to provide redundant connectivity between all locations. The following diagram is an example deployment spanning multiple locations.

Figure 3.7. Multiple Interconnected Routers



Revised on 2018-07-16 16:08:02 EDT