Red Hat JBoss AMQ 7.0

Introducing Red Hat JBoss AMQ 7

Overview of Features and Components

Last Updated: 2017-09-01

Red Hat JBoss AMQ 7.0 Introducing Red Hat JBoss AMQ 7

Overview of Features and Components

Legal Notice

Abstract

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

Table of Contents

CHAPTER 1. ABOUT RED HAT JBOSS AMQ 7.0

Red Hat JBoss AMQ is a fast and flexible messaging platform that easily integrates the various components in your application environments and efficiently scales to meet their workload needs. The components of AMQ can be deployed and managed in a variety of configurations. AMQ Broker supports the multiple languages, protocols, and platforms found within the growing collection of AMQ Clients, for example. New to Red Hat JBoss AMQ 7.0, the AMQ Interconnect router uses the AMQP protocol to distribute and scale your messaging resources across your network like never before.

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

1.1. WHAT'S NEW

In addition to the new AMQ Interconnect router, Red Hat JBoss AMQ 7.0 includes a number of enhancements to the AMQ Broker. It also adds support for new client libraries.

AMQ Broker

AMQ Broker uses a new, high-performance messaging implementation based on ActiveMQ Artemis, including the use of an asynchronous journal to improve the messaging store.

AMQ Interconnect

Now included as part of Red Hat JBoss AMQ 7.0, AMQ Interconnect is a message router you can use to build large-scale messaging networks. AMQ Interconnect uses the AMQP protocol to create a redundant application-level messaging network. It works as an interconnect layer between messaging clients and their brokers.

New AMQ Clients

Red Hat JBoss AMQ 7.0 expands its support for popular messaging APIs and protocols by adding new client libraries, including the following clients.

- AMQ C++
  A flexible and capable AMQP 1.0 C++ messaging library. Its event-driven API simplifies integration with existing applications. It can be used to create not only messaging clients but brokers, routers, bridges, and proxies.

- AMQ JavaScript
  An AMQP 1.0 JavaScript messaging library for use in Node.js and browsers. It shares the programming model and capabilities of the AMQP C++ Client.

1.2. KEY FEATURES

Red Hat JBoss AMQ 7.0 components enable developers to build flexible and reliable messaging applications that are highly performant, highly available, and easily administered. Key features include:

Messaging at Internet Scale

Red Hat JBoss AMQ 7.0 gives you the tools to build a seamless worldwide network. It connects clients, brokers, and standalone services to build an advanced messaging fabric.

Top-Tier Security and Performance

Modern SSL/TLS encryption and extensible SASL authentication. Red Hat JBoss AMQ 7.0 delivers fast, high-volume messaging and class-leading JMS performance.

Broad Platform and Language Support

Components for multiple languages and operating systems, so your diverse application components can communicate. Red Hat JBoss AMQ 7.0 supports C++, Java, Python, and .NET applications, as well as Linux, Windows, and JVM-based environments.

Focused on Standards

Red Hat JBoss AMQ 7.0 implements the Java JMS 1.1 and 2.0 API specifications. Its components support the ISO-standard AMQP 1.0 message protocol and the MQTT, STOMP, and WebSocket protocols.

Core Services to Speed Application Development

Included with every subscription is the JBoss Core Services Collection , a set of popular components that provide load balancing, system management, single sign-on, API management, and more.

Centralized Management

Red Hat JBoss AMQ 7.0 allows you to administer all AMQ components from a single management interface. You can use JMX to remotely invoke management operations as well.

CHAPTER 2. COMPONENT OVERVIEW

Red Hat JBoss AMQ 7.0 consists of the AMQ Broker, the AMQ Clients, and a new dispatch router called AMQ Interconnect. They all work to enable remote communication among distributed client applications. Red Hat JBoss AMQ 7.0 can communicate with a wide variety of JMS and non-JMS clients. It also supports various protocols such as AMQP and STOMP, among others. The AMQ Console is the single interface used to configure, deploy and manage AMQ components.

Because of the many components it makes available for deployment and use, Red Hat JBoss AMQ 7.0 is more like a toolbox than a monolithic application platform. In the toolbox are 4 major tools:

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

2.1. AMQ BROKER

AMQ Broker is a full-featured, message-oriented middleware broker. It offers specialized queueing behaviors, message persistence, and manageability. Core messaging is provided by Apache ActiveMQ with support for different messaging styles such as publish-subscribe, point-to-point, and store-and-forward. It supports multiple protocols and client languages, freeing you to use many if not all of your application assets. Lastly, AMQ Broker is supported to work with Red Hat JBoss Enterprise Application Platform.

For more information on the AMQ Broker see Using AMQ Broker.

2.2. AMQ INTERCONNECT

AMQ Interconnect, a new component of AMQ 7, provides flexible routing of messages between any AMQP-enabled endpoints, whether they are clients, servers, brokers, or any other entity that can send or receive standard AMQP messages. For example, a messaging client can make a single AMQP connection into a messaging bus built of AMQ Interconnect routers. Then, using that one connection, the client can exchange messages both with brokers and with other endpoints without involving any broker at all.

AMQ Interconnect does not need to use clustering for high availability. It is meant to be deployed in topologies of multiple routers, preferably with redundant paths, which it uses to provide continued connectivity. Consequently, routing networks should scale much more easily when using AMQ Interconnect because you can distribute your network processing resources.

For more information on AMQ Interconnect see Using AMQ Interconnect.

2.3. AMQ CLIENTS

AMQ emphasizes fast and reliable communication between clients and servers. It supports many well-known languages and protocols including those used by the following clients.

- Using the AMQ JMS Client
- Using the AMQ C++ Client

- Using the AMQ Python Client
- Using the AMQ .NET Client
- Using the AMQ JavaScript Client

2.4. AMQ CONSOLE

AMQ Console is a management tool for administering AMQ brokers and routers in a single convenient interface. AMQ Console helps you maintain and grow your AMQ environments, including managing destinations, connections, and other administrative objects. You can, for example, launch into a mode that displays the command console, or you can use a remote AMQ Console to connect to a broker on your network. Conveniently, the AMQ Console includes REST and JMX interfaces for remote management.

2.5. COMPONENT COMPATIBILITY

The following table lists the supported languages, platforms, and protocols of Red Hat JBoss AMQ 7.0 components. Note that components sharing an AMQP version can interoperate. For instance, although the AMQ Broker is implemented in Java, it does not require Java clients for communication.

Table 2.1. AMQ Component Compatibility

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

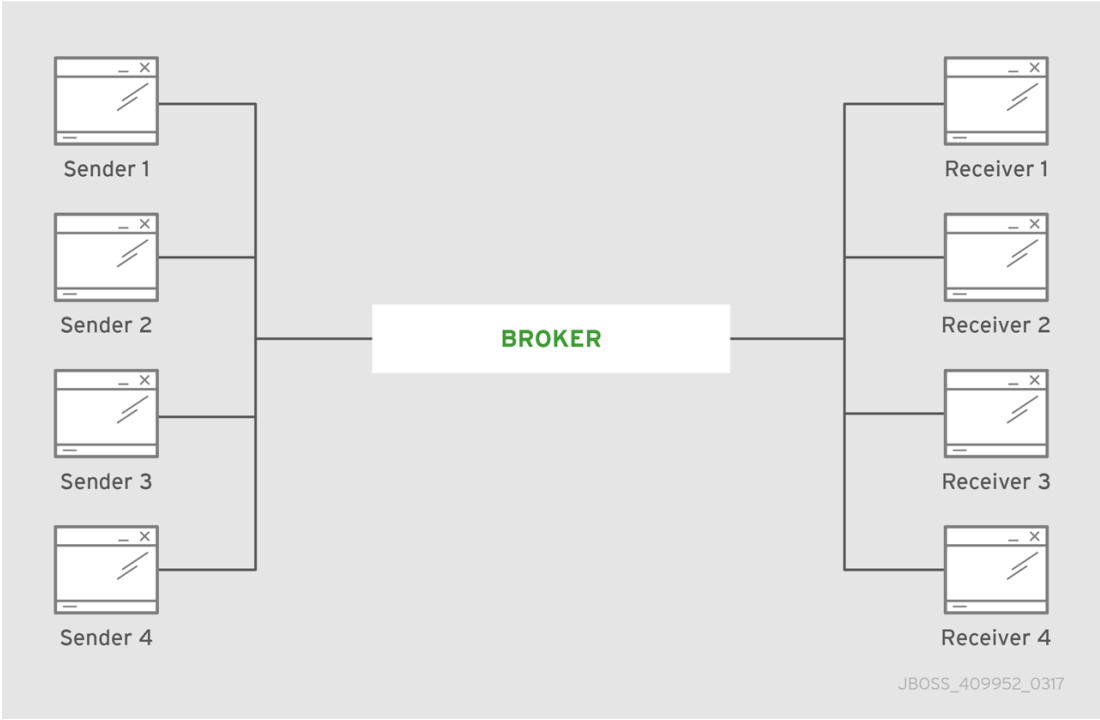
CHAPTER 3. COMMON DEPLOYMENT PATTERNS

Red Hat JBoss AMQ 7.0 can be set up in a large variety of topologies. The following are just some of the common deployment patterns you can implement using Red Hat JBoss AMQ 7.0 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, since 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*, where the central broker becomes the hub and the clients the spokes.

Figure 3.1. Central Broker Pattern

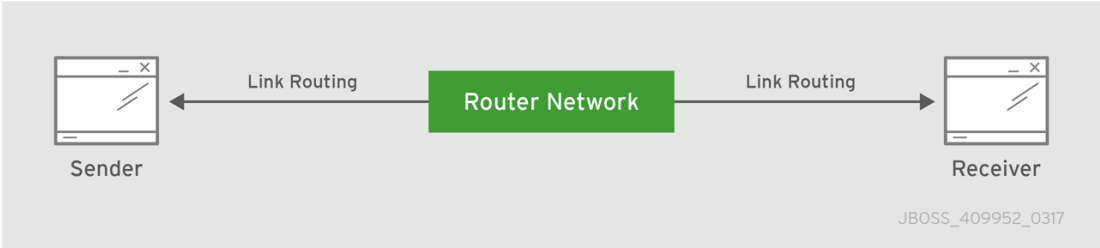


The only critical element is the central broker node. You would need to focus your maintenance efforts on keeping this broker up and running.

3.2. ROUTED MESSAGING

Sometimes an application requires sending request and response messages in real time. In these cases, having a broker in the middle that stores and forwards messages as it routes them is unnecessary and costly. With Red Hat JBoss AMQ 7.0 you can use a router in place of a broker to avoid such costs. The router does not store messages before forwarding them to a destination and works as a lightweight conduit that directly connects two endpoints.

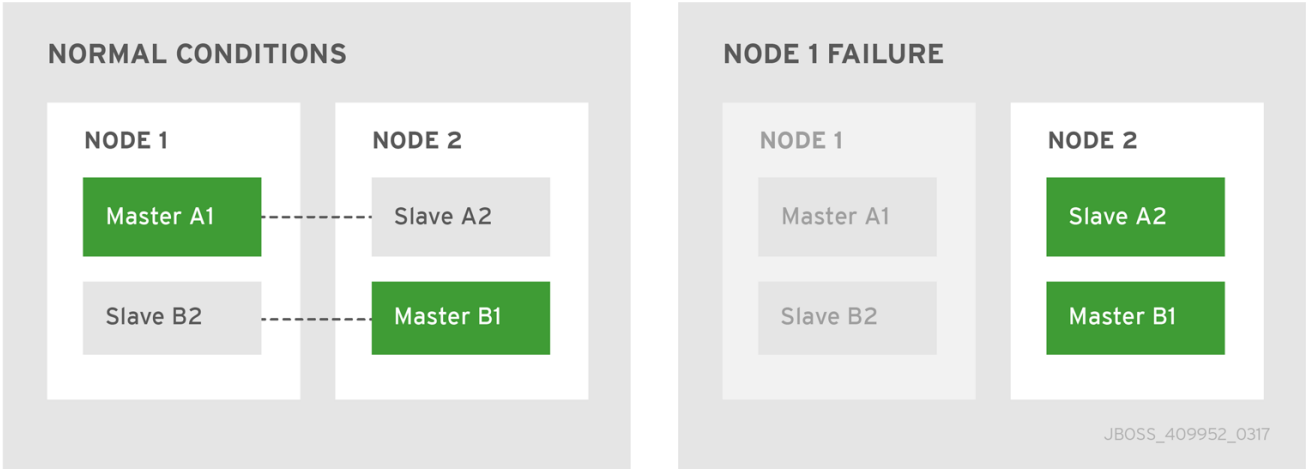
Figure 3.2. Brokerless Router Pattern



3.3. HIGHLY AVAILABLE BROKERS

To ensure brokers are up and running for their clients, deploy a highly available (HA) master-slave pair to create a backup group. You could, 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.

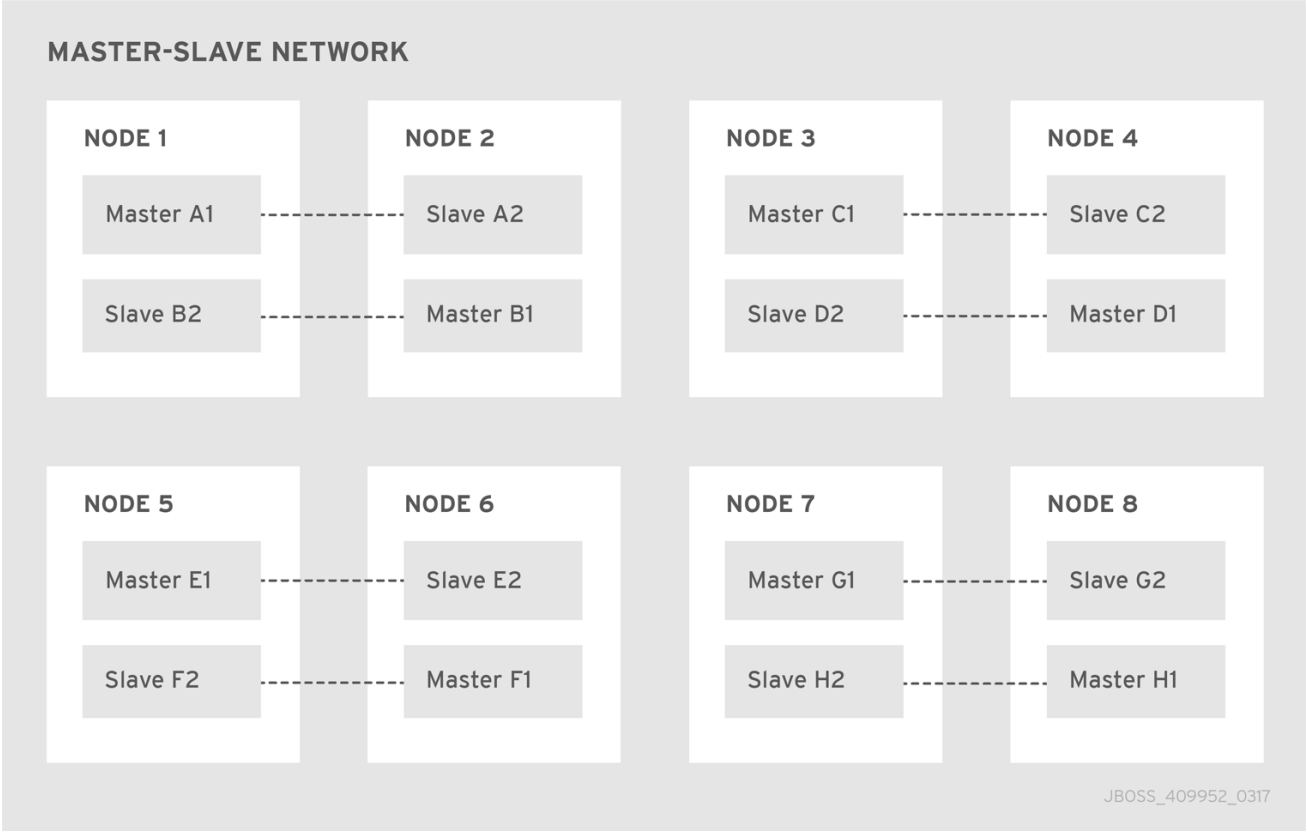
Figure 3.3. Master-slave Pair



Under normal operating conditions, one master broker is active on each node, here defined as 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.

Using the basic principles of the master-slave group, you could scale up to 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 diagram below consists of eight master-slave groups distributed over eight nodes.

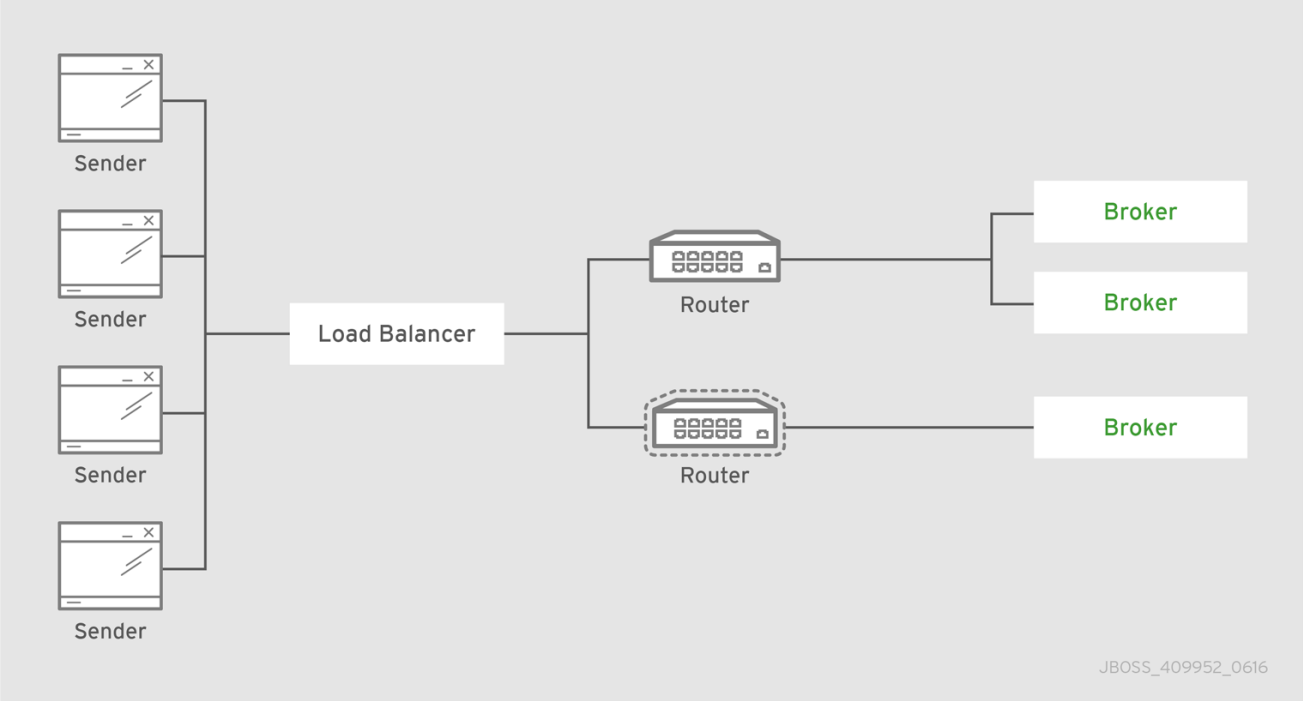
Figure 3.4. Master-slave Network



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

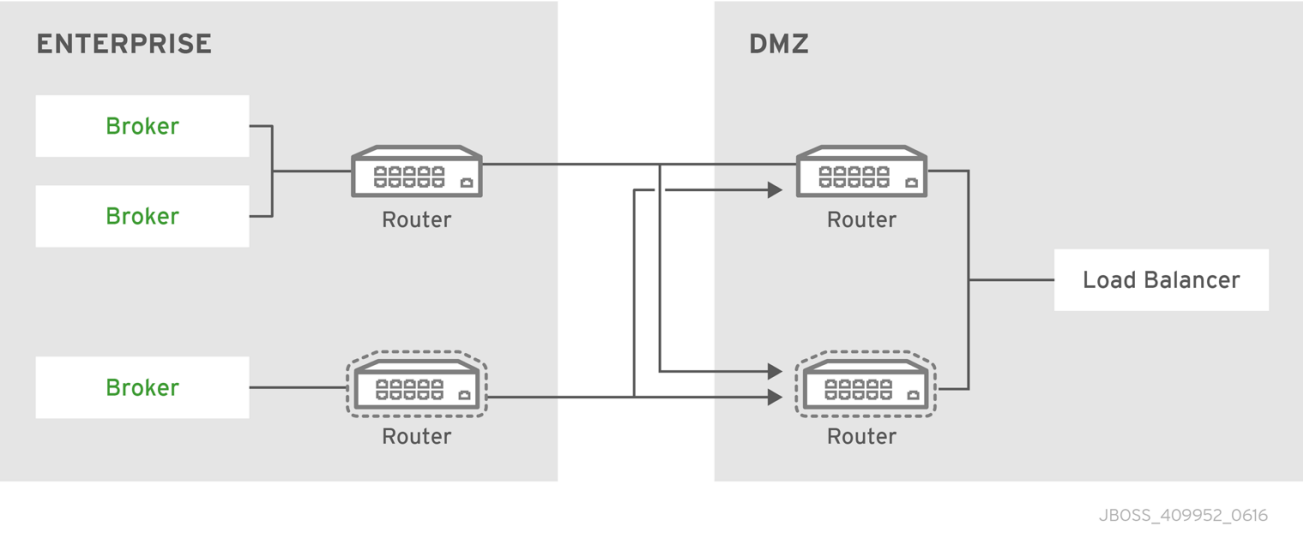


For even greater scalability, you could use a larger number of routers, three or four for example. Each router would be connected directly to all of the others.

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



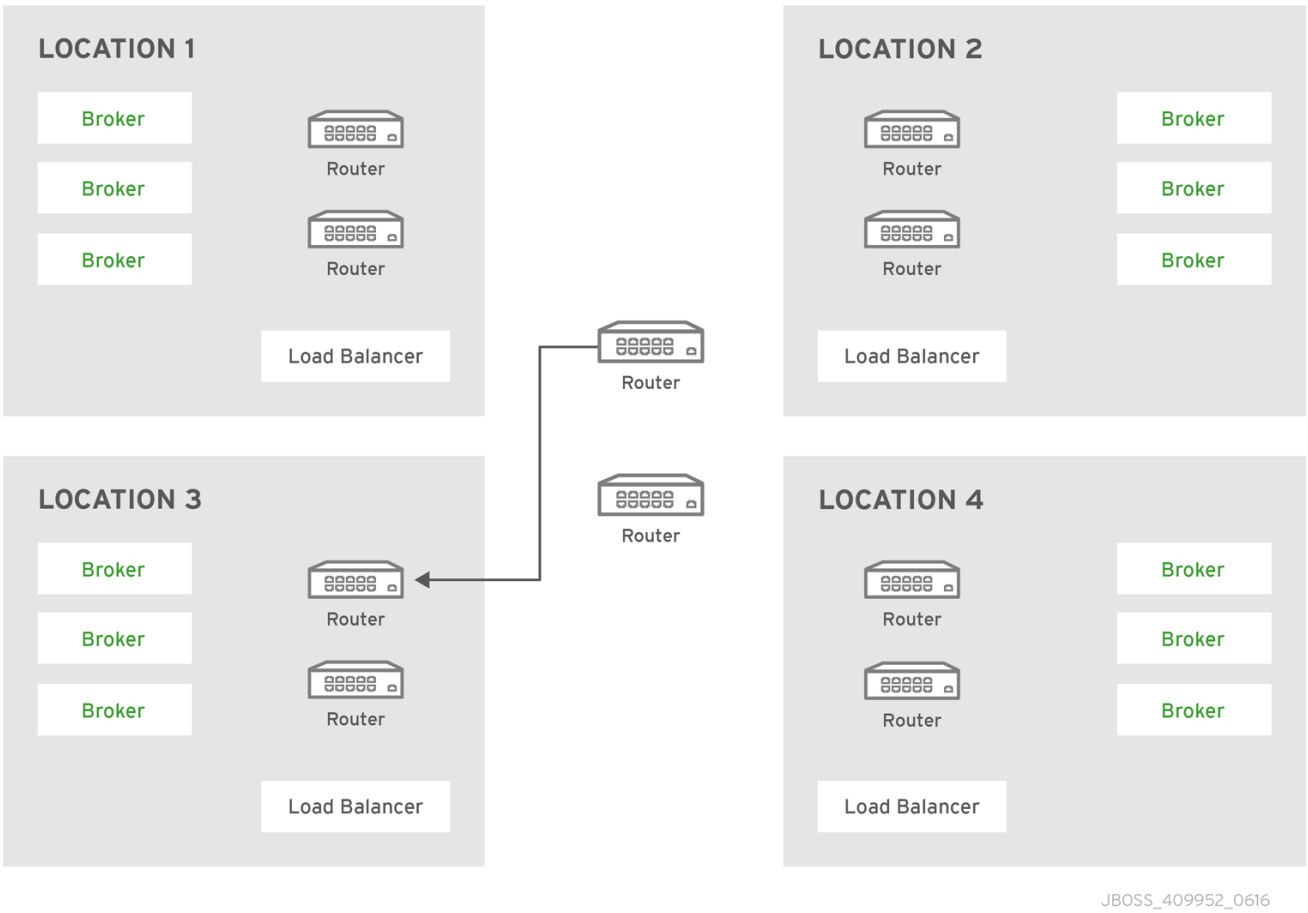
Important notes about the DMZ topology:

- 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, while 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 AMQP connections are permitted from the outside into the enterprise. The AMQP protocol permits bi-directional communication once a connection is established, however.

3.6. ROUTER PAIRS IN DIFFERENT DATA CENTERS

You can use a more complex topology in a deployment of Red Hat JBoss AMQ 7.0 components that spans multiple locations. You could, 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 the four locations. Below is a diagram of an example deployment that spans multiple locations.

Figure 3.7. Multiple Interconnected Routers



Revised on 2017-09-01 11:04:07 EDT