◆ ◆ ◆ CHAPTER 4

Broker Clusters

Message Queue supports the use of *broker clusters*: groups of brokers working together to provide message delivery services to clients. Clusters enable a Message Queue service to scale messaging operations by distributing client connections among multiple brokers. Because a cluster consists of multiple brokers, the cluster helps protect against individual broker failure. Two cluster models provide different levels of message service availability.

This chapter discusses the architecture and internal functioning of broker clusters. It covers the following topics:

Cluster Models

Message Queue supports two clustering models both of which provide a scalable message service, but with each providing a different level of message service availability:

- **Conventional broker clusters.** A conventional broker cluster provides for *service availability*. When a broker or a connection fails, clients connected to the failed broker reconnect to another broker in the cluster. However, messages and state information stored in the failed broker cannot be recovered until the failed broker is brought back online. The broker or connection failure can therefore result in a significant delay and in messages being delivered out of order.
- **Enhanced broker clusters.** An enhanced broker cluster provides for *data availability* in addition to service availability. When a broker or a connection fails, another broker takes over the pending work of the failed broker. The failover broker has access to the failed broker's messages and state information. Clients connected to the failed broker reconnect to

the failover broker. In an enhanced cluster, as compared to a conventional cluster, a broker or connection failure rarely results in significant delays in message delivery and messages are always delivered in order.

Note – Despite the message service availability offered by both conventional and enhanced broker clusters, they do not provide a guarantee against failure and the possibility that certain failures, for example in the middle of a transaction, could require that some operations be repeated. It is the responsibility of the messaging application (both producers and consumers) to handle and respond appropriately to failure notifications from the messaging service.

Conventional and enhanced broker clusters are built on the same underlying infrastructure and message delivery mechanisms. They differ in how brokers in the cluster are synchronized with one another and in how the cluster detects and responds to failures.

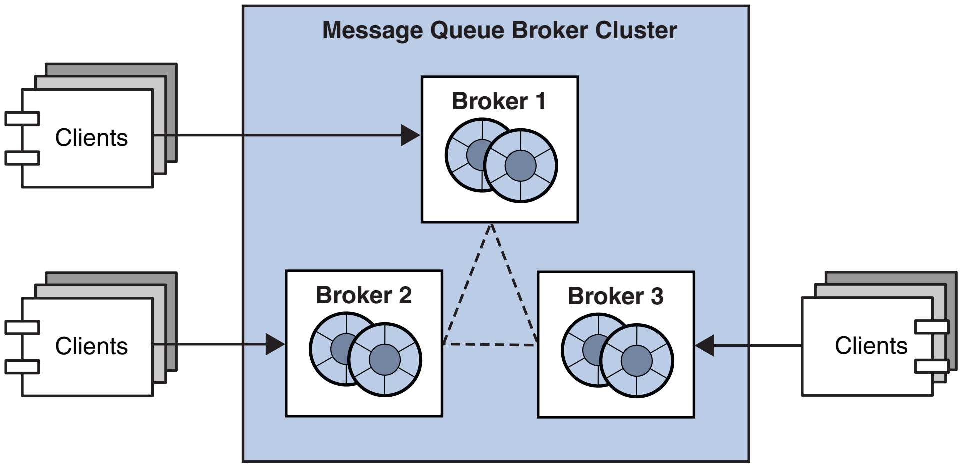
The sections that follow first describe the infrastructure and delivery mechanisms common to both clustering models, after which the unique aspects of each model is explained.

Cluster Message Delivery

A broker cluster facilitates the delivery of messages between client applications that are connected to different brokers in the cluster.

The following illustration shows salient features of a Message Queue broker cluster. Each of three brokers is connected to the other brokers in the cluster: the cluster is fully-connected. The brokers communicate with each other and pass messages by way of a special *cluster connection service*, shown in Figure 4–1 by the dashed lines.

FIGURE 4–1 Message Queue Broker Cluster



Each broker typically has a set of messaging clients (producers and/or consumers) that are directly connected to that broker. For these client applications, the broker to which they are directly connected is called their home broker. Each client communicates directly only with its home broker, sending and receiving messages as if that broker were the only broker in the cluster.

Accordingly, a producer in the cluster produces messages to a destination in its home broker. The home broker is responsible for routing and delivering the messages to all consumers of the destination, whether these consumers are local (connected to the home broker) or remote (connected to other brokers in the cluster). The home broker works in concert with the other brokers to deliver messages to all consumers, no matter what brokers they are connected to.

Propagation of Information Across a Cluster

To facilitate delivery of messages across the cluster, information about the destinations and consumers of each broker is propagated to all brokers in the cluster. Each broker therefore stores the following information:

- The name, type, and properties of all physical destinations in the cluster
- The name, location, and destination of interest of each message consumer

Changes in this information are propagated whenever one of the following events occurs:

- A destination on one of the cluster's brokers is created or destroyed.

  There are minor variations in the propagation of destinations, depending on the kind of destination:

  - Admin-created destinations. When the destination is created, it is propagated across the cluster. When the destination is deleted on any broker in the cluster, it's deletion is propagated across the cluster.
  - Auto-created destinations. When a producer is created and the corresponding destination does not exist, the destination is auto-created on the producer's home broker, but is not immediately propagated across the cluster. By contrast, when a consumer is created and the corresponding destination does not exist, the destination is auto-created on the consumer's home broker *and* is propagated across the cluster (as part of the propagation of information about the consumer). An auto-created destination can be explicitly deleted by an administrator on each broker. Otherwise, the destination will be automatically deleted on each broker either when it has had no consumers and has contained no messages for two minutes, or when the broker restarts and there are no messages in the destination.
  - Temporary destinations. When the destination is programmatically created, it is propagated across the cluster. If the consumer of the temporary destination is set to automatically reconnect in the event of failure, then the destination is stored persistently, and propagated across the cluster as a persistent destination. When the consumer connection to the temporary destination closes, the destination is deleted, and it's deletion is propagated across the cluster. If the home broker of the consumer of a persistent temporary destination fails and is restarted, and if the consumer does not reconnect within a specific time interval, then it is assumed that the consumer has failed and the temporary destination is deleted, and it's deletion is propagated across the cluster.
- The properties of a destination are changed.
- A message consumer is registered with its home broker.
- A message consumer is disconnected from its home broker (whether explicitly or through failure of the client, the broker, or the network).

The propagation of destination and consumer information across the cluster means that destinations and consumers are essentially global to the cluster. In the case of destinations, properties set for a physical destination (see “Configuring Physical Destinations” on page 67) apply to all instances of that destination in the cluster. Distributing producers across a cluster thus results in cumulative cluster-wide limits specified by destination properties such as the maximum number of messages, the maximum number of message bytes, and the maximum number of producers.

Message Delivery Mechanisms

Despite the global nature of destinations and consumers in a cluster, a home broker has special responsibilities with respect to both its producers and consumers:

- A producer’s home broker is responsible for persisting and routing messages originating from that producer, for logging, for managing transactions, and for processing acknowledgements from consuming clients across the cluster.
- A consumer’s home broker is responsible for persisting information about consumers, for delivering remotely produced messages to the consumer, for letting a producer’s home broker know whether the consumer is still available, and for letting a producer's home broker know when each message has been successfully consumed.

The cluster connection service transports payload messages, when needed, from destinations on a home broker to destinations on remote brokers. It also transports control messages, such as client acknowledgements, from remote brokers back to a home broker. The cluster attempts to minimize message traffic across the cluster. For example, it only sends a message to a remote broker if the remote broker is home to a consumer of the message. If a remote broker has two identical consumers for the same destination (for example two topic subscribers), the message is sent over the wire only once. (You can further reduce traffic by setting a destination property specifying that delivery to local consumers has priority over delivery to remote consumers.)

If secure message delivery is required, you can configure a cluster to also provide secure, encrypted delivery of messages between brokers.

As a result of the cluster delivery mechanisms described above, each broker in a cluster stores different persistent messages and maintains different state information. If a broker fails, the mechanisms for recovering its persistent information depends on the cluster model being used, as described in subsequent sections.

Conventional Clusters

Message Queue supports two types of conventional clusters:

- Conventional cluster with master broker
- Conventional cluster of peer brokers

Both of these types provide service availability in the same way, but they differ in the way that cluster state information is maintained.

The following figures illustrate the two types of conventional broker cluster.

FIGURE 4–2 Conventional Broker Cluster with Master Broker

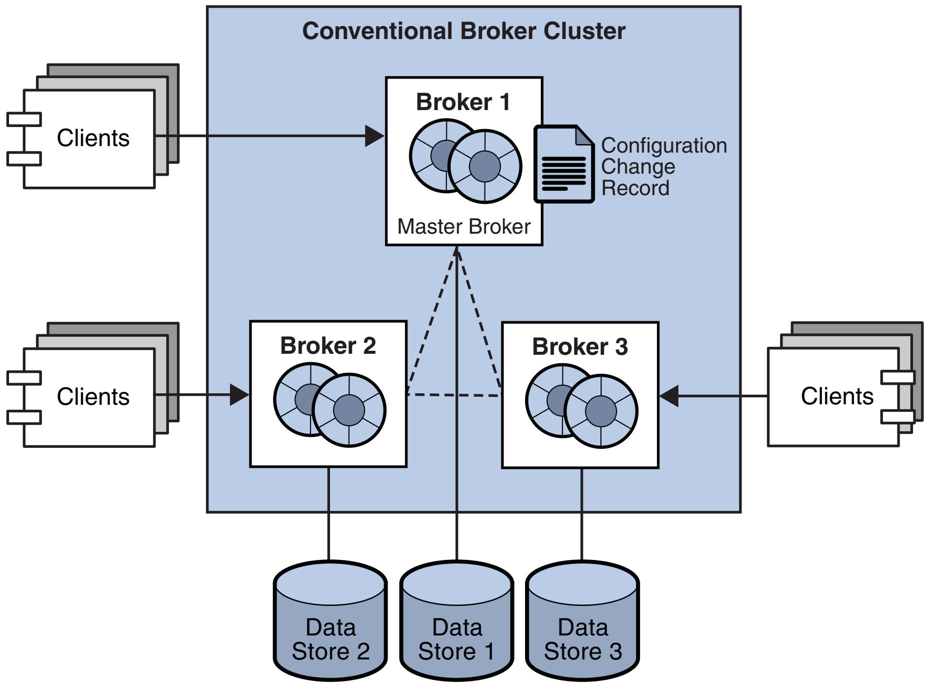
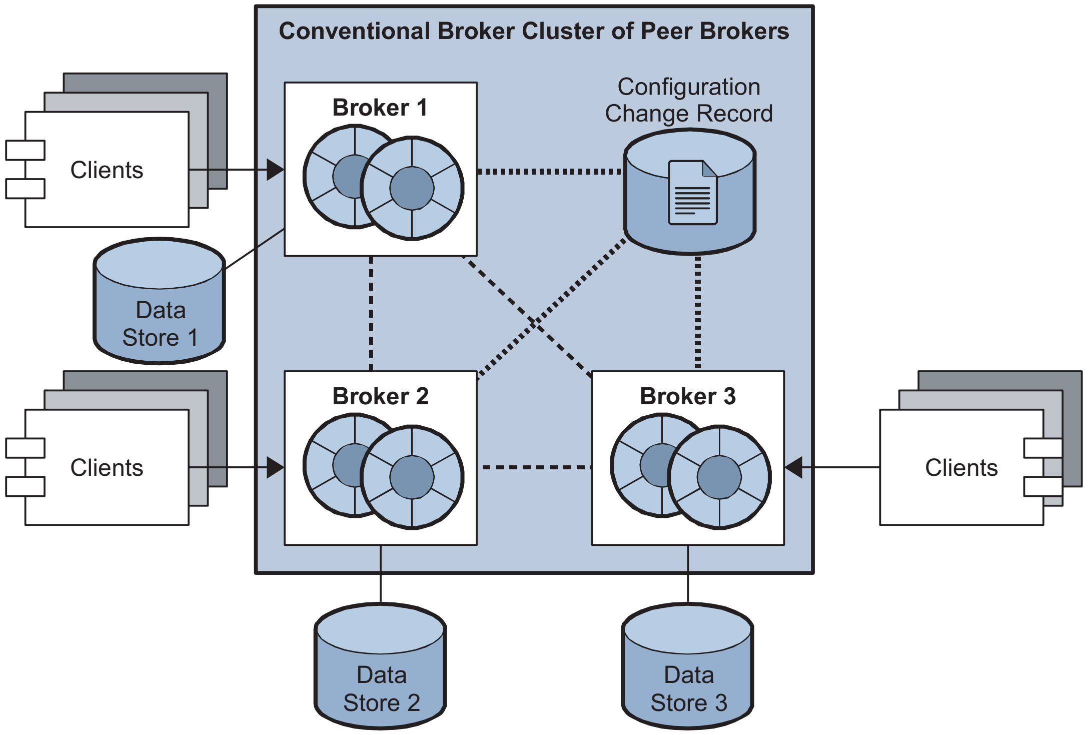


FIGURE 4–3 Conventional Broker Cluster of Peer Brokers



Conventional broker clusters have the following characteristics:

- **Data Synchronization**

  Each broker has its own respective persistent data store in which destinations, persistent messages, and other state information is stored. Some of this information (for example, destinations and durable subscriptions) has been propagated to the broker from other brokers in the cluster. If a broker fails, it is possible for this information to become out of sync with the information stored by other brokers in the cluster. To guard against this possibility in a conventional broker cluster, a *configuration change record* is maintained to track changes to the cluster's propagated persistent entities. In a conventional cluster with master broker, one broker, designated as the *master broker*, maintains the configuration change record. In a conventional cluster of peer brokers, the configuration change record is maintained in a JDBC data store that is accessible to all the brokers.

  When an offline broker comes back online (or when a new broker is added to the cluster), it consults the configuration change record for information about destinations and durable subscribers, then exchanges information with other brokers about its currently active message consumers.

In a conventional cluster with master broker, the master broker should always be the first broker started within the cluster because other brokers cannot complete their initialization without accessing the configuration change record. Furthermore, if the master broker goes offline, destination and durable subscriber information cannot be propagated across the cluster. Under these conditions, you get an exception if you try to create, reconfigure, or destroy a destination or a durable subscription (auto-created destinations and temporary destinations are not affected), or attempt a related operation. Similarly, in the absence of a master broker, any client application attempting to create a durable subscriber or unsubscribe from a durable subscription gets an error. Nevertheless, client applications can successfully interact with an existing durable subscriber.

Message production, delivery, and consumption can continue uninterrupted without a master broker.

- **Failure Detection and Recovery**

  A conventional broker cluster detects failures when one broker tries to send data to another broker and an exception is thrown. When the cluster encounters a failed connection between brokers, it cannot do anything to recover, other than stop sendng data. It is the responsibility of an administrator to monitor brokers in the cluster by using Message Queue administration tools (see "Administration Tools" on page 78) and perform the appropriate recovery operations.

- **Client Reconnect**

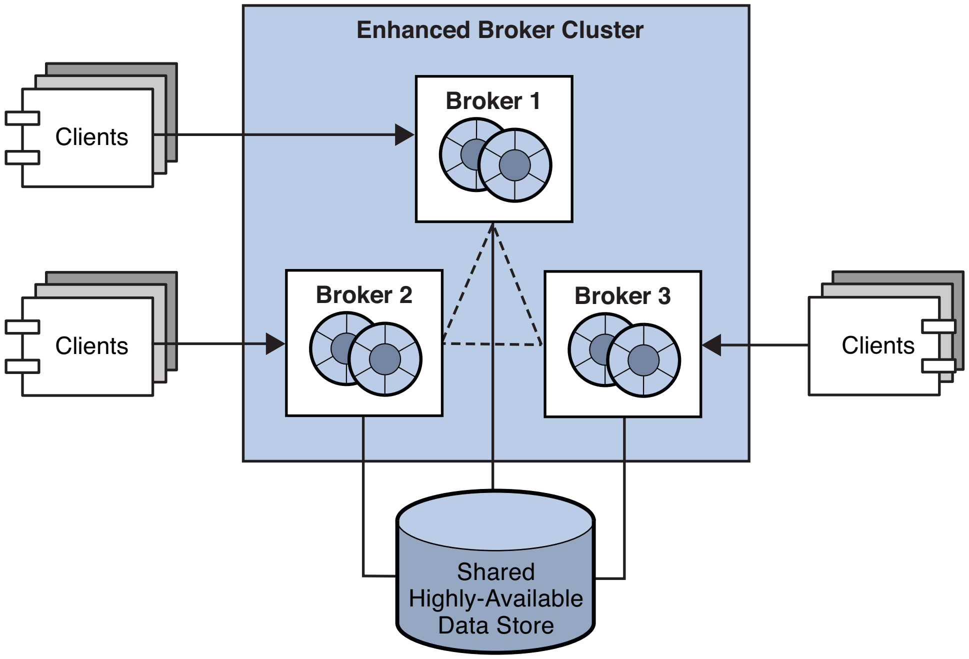
  If a broker or its connection to a client fails, the client automatically attempts to reconnect to the same or another broker in the cluster. The reconnect is governed by connection properties that specify the order and frequency by which the client attempts to reconnect to brokers in the cluster. The broker to which the client successfully reconnects becomes the client's new home broker.

  In this scenario, the new home broker (if different from the failed broker) does not have all the client-related state information that was previously held by the failed broker. For example, messages to have been consumed by the client or the state of transactions involving the client might have been lost. As a result, the failure of a broker in a conventional cluster can cause a delay in message delivery (until the failed broker restarts and the client reconnects).

Enhanced Clusters

The following figure illustrates an enhanced broker cluster. An enhanced broker cluster provides both service availability and data availability.

FIGURE 4–4 Enhanced Cluster



An enhanced broker cluster has the following characteristics:

- **Data Synchronization**

  All brokers in an enhanced cluster share a common persistent data store in which destinations, persistent messages, and other state information is stored for each broker. Because all brokers share the same data store, each broker is able to access the state information stored by other brokers in the cluster. When a broker that has been offline rejoins the cluster (or when a new broker is added to the cluster) it is able to access the most current information simply by accessing the shared data store. Similarly, if a broker fails, another broker is able to access and take over the failed broker's information in the shared data store.

  To achieve data availability, the shared data store must be a highly-available JDBC database. While it is possible to use a shared data store that is not highly-available, such a data store would represent a single point of failure for the cluster, and pose a normally unacceptable risk for a production message service: all brokers in the cluster would be impacted if the shared data store were to become unavailable.

- **Failure Detection and Recovery**

  An enhanced cluster makes use of a distributed heartbeat service by which brokers inform other brokers that they are online and accessible by the cluster connection service. The heartbeat service also updates broker state information in the cluster's shared data store. When no heartbeat packet is detected from a broker for a configurable number of heartbeat intervals, the broker is considered suspect of failure. The other brokers in the cluster then begin to monitor the suspect broker's state information in the shared data store to confirm whether the broker is still online. If the suspect broker does not update its state information within a configurable interval, it is considered to have failed. There is a trade-off between the speed and the accuracy of failure detection: configuring the cluster for quick failure detection increases the likelihood that a slow broker will erroneously be considered to have failed.

If these failure detection services operating in tandem determine that a broker has failed, then a failover broker is selected from among the remaining online brokers to take over the pending work of the failed broker.

The failover broker attempts to take over the failed broker's persistent state (pending messages, destinations, durable subscriptions, pending acknowledgments, and open transactions) so as to provide uninterrupted service to the failed broker's clients. If two or more brokers attempt such a takeover, only the first will succeed (the first acquires a lock on the failed broker's data in the shared data store, preventing subsequent takeover attempts).

The takeover of a failed broker's state happens very rapidly, however while in process, the failover broker cannot accept new client connections.

Once takeover is complete and a period for clients to reconnect to the failover broker has elapsed, the failover broker will clean up any transient resources (such as transactions and temporary destinations) belonging to the failed broker.

- **Client Reconnect**

  If a broker fails, its clients automatically reconnect to the failover broker, which becomes their new home broker. The reconnect process is a dynamic interplay between the client runtime and the broker cluster: if a client attempts to reconnect to a broker that is not the failover broker, the reconnect is rejected and the client is redirected to the failover broker.

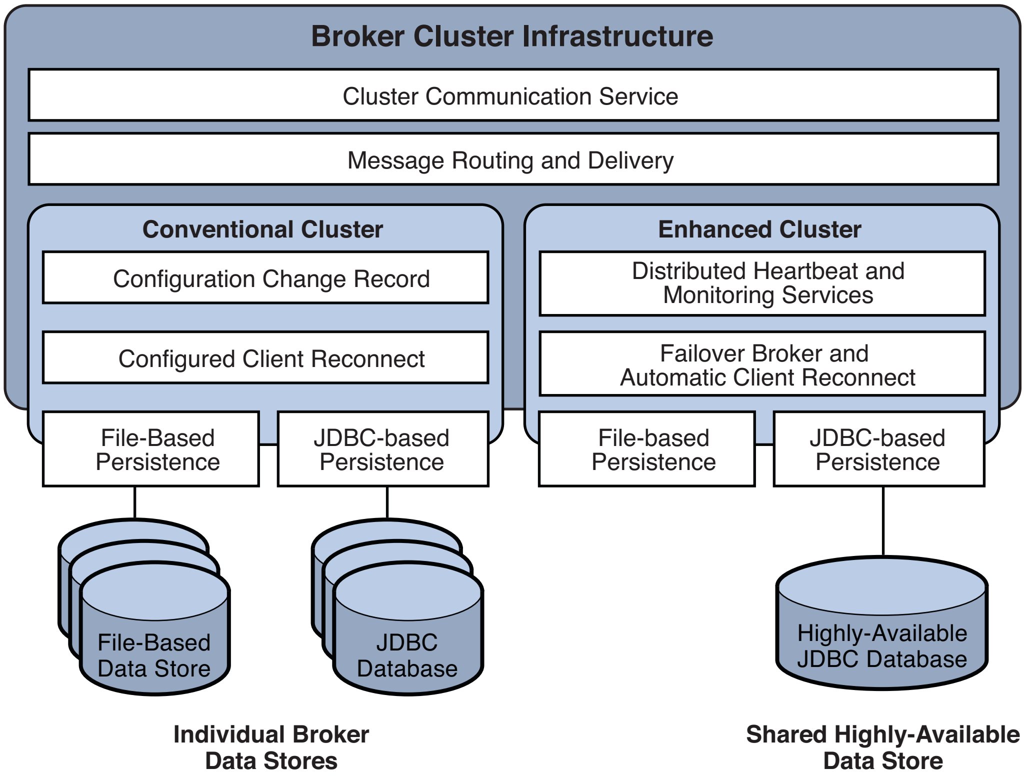
  In this scenario, the new home broker (the failover broker) has immediate access to all the client-related state information that was previously held by the failed broker. The failover broker can therefore take over where the failed broker left off. As a result, the failure of a broker in an enhanced cluster will not cause a failure in message delivery. However, during the short time required for takeover to complete, the failover broker cannot accept new client connections, causing a short delay in client reconnects, and a corresponding short delay in message delivery.

To configure an enhanced cluster you set cluster configuration properties for each broker in the cluster. These properties are detailed in "Enhanced Broker Cluster Properties" in *Open Message Queue 4.5 Administration Guide*.

Cluster Models Compared

Conventional and enhanced cluster models share the same basic infrastructure. They both use the cluster communication service to enable message delivery between producers and consumers across the cluster. However, as shown in the following figure and described in previous sections, these models differ in how destination and consumer information is synchronized across the cluster, in the mechanisms for detecting failure, in how client reconnect takes place.

FIGURE 4–5 Cluster Infrastructure


Broker Cluster Infrastructure
Cluster Communication Service
Message Routing and Delivery
Conventional Cluster
Configuration Change Record
Configured Client Reconnect
File-Based Persistence
JDBC-based Persistence
Enhanced Cluster
Distributed Heartbeat and Monitoring Services
Failover Broker and Automatic Client Reconnect
File-based Persistence
JDBC-based Persistence
File-Based Data Store
JDBC Database
Highly-Available JDBC Database
Individual Broker Data Stores
Shared Highly-Available Data Store

In addition, while both models rely on the broker's persistence interfaces (both flat-file and JDBC), in the case of enhanced clusters the shared data store must be a highly-available JDBC database (a highly-available file-based data store has not yet been implemented).

The following table summarizes the functional differences between the two cluster models. This information might help in deciding which model to use or whether to switch from one to another.

TABLE 4–1 Clustering Model Differences

Functionality	Conventional	Enhanced
Performance	Faster than enhanced cluster model.	Slower than conventional cluster model.
Service availability	Yes, but some operations are not possible if master broker is down.	Yes.
Data availability	No. State information in failed broker is not available until broker restarts.	Yes at all times.
Transparent recovery from failure	No. Message delivery is interrupted. Also, client reconnects might not be possible if failure occurs during a transaction commit (rare).	Yes. No interruption in message delivery. If failure occurs during a transaction commit, an exception might be thrown indicating that the transaction could not be committed (extremely rare).
Configuration	Set appropriate cluster configuration properties for each broker.	Set appropriate cluster configuration properties for each broker.
Additional requirements	None.	Highly-available database.
Restricted to subnet	No.	Yes.

Cluster Configuration

Depending on the clustering model used, you must specify appropriate broker properties to enable the Message Queue service to manage the cluster. This information is specified by a set of *cluster configuration properties*,. Some of these properties must have the same value for all brokers in a cluster; others must be specified for each broker individually. It is recommended that you place all configuration properties that must be the same for all brokers in one central *cluster configuration file* that is referenced by each broker at startup time. This ensures that all brokers share the same common cluster configuration information.

See "Configuring Broker Clusters" in *Open Message Queue 4.5 Administration Guide*for detailed information on cluster configuration properties.

Note – Although the cluster configuration file was originally intended for configuring clusters, it is also a convenient place to store other (non-cluster-related) properties that are shared by all brokers in a cluster.

For complete information about administering broker clusters, see Chapter 10, "Configuring and Managing Broker Clusters," in *Open Message Queue 4.5 Administration Guide*. For information about the effect of reconnection on the client, see "Connection Event Notification" in *Open Message Queue 4.5 Developer's Guide for Java Clients* and "Client Connection Failover (Auto-Reconnect)" in *Open Message Queue 4.5 Developer's Guide for Java Clients*.