Oracle Fusion Middleware

Developing Message-Driven Beans for Oracle WebLogic Server



26 (26.1.0.0.0)
G56957-01
November 2026



Oracle Fusion Middleware Developing Message-Driven Beans for Oracle WebLogic Server, 26 (26.1.0.0.0)

G56957-01

Copyright © 2007, 2026, Oracle and/or its affiliates.

Primary Author: Oracle Corporation

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software, software documentation, data (as defined in the Federal Acquisition Regulation), or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software," "commercial computer software documentation," or "limited rights data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle®, Java, MySQL, and NetSuite are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Oracle customer access to and use of Oracle support services will be pursuant to the terms and conditions specified in their Oracle order for the applicable services.

Contents

Preface

Audience i
Documentation Accessibility i
Diversity and Inclusion i
Related Resources i
Conventions i

1 Understanding Message-driven Beans

JCA-Based MDBs 1

2 MDB Life Cycle

Overview 1
MDBs and Concurrent Processing 1
Limitations for Multi-threaded Topic MDBs 2

3 MDBs and Messaging Models

Point-to-Point (Queue) Model 1
Publish/Subscribe (Topic) Model 2
Exactly-Once Processing 3

4 Deploying MDBs

Destination and MDBs: Collocation vs. non-Collocation 1
Collocated Destination/MDBs 1
Non-Collocated Destination/MDBs 3
JMS Distributed Destinations 4
Best Practice 5

5 Programming and Configuring MDBs: Main Steps

Required JMS Configuration 1

Copyright © 2007, 2026, Oracle and/or its affiliates.

Create MDB Class and Configure Deployment Elements 2

6 Programming and Configuring MDBs: Details

Configuring Destination Type 1
Configuring Transaction Management Strategy for an MDB 2
Configuring MDBs for Destinations 2
Whether to Use Foreign JMS Server Mappings 3
How to Set provider-url 4
How to Set initial-context-factory 4
How to Set destination-jndi-name 4
How to Set connection-factory-jndi-name 4
Common Destination Scenarios: Illustrations and Key Element Settings 5
Configuring Message Handling Behaviors 10
Ensuring Message Receipt Order 10
Preventing and Handling Duplicate Messages 11
Redelivery and Exception Handling 12
Using the Message-Driven Bean Context 13
Configuring Suspension of Message Delivery During JMS Resource Outages 13
Configuring the Number of Seconds to Suspend a JMS Connection 14
How the EJB Container Determines How Long to Suspend a JMS Connection 14
Turning Off Suspension of a JMS Connection 14
Configuring a Security Identity for a Message-Driven Bean 15
Using MDBs With Cross Domain Security 15
Configuring EJBs to Use Logical Message Destinations 16
Declaring and Mapping a Logical JMS Destination 17
Configuring a Message Destination Reference for Enterprise Bean Code 17
Linking an MDB to a Logical JMS Destination 17
Using Application-Scoped JMS Resources 18

7 Using EJB 4.0 Compliant MDBs

Implementing EJB 4.0 Compliant MDBs 1
Programming EJB 4.0 Compliant MDBs 1
MDB Sample Using Annotations 3

8 Migration and Recovery for Clustered MDBs

9 Using Batching with Message-Driven Beans

Configuring MDB Transaction Batching 1

Copyright © 2007, 2026, Oracle and/or its affiliates.

How MDB Transaction Batching Works 2

10 Configuring and Deploying MDBs Using JMS Topics

Supported Topic Types 1
The Most Commonly Used MDB Attributes 2
Setting the JMS Destination, Destination Type, and Connection Factory 3
Setting Subscription Durability 3
Setting Automatic Deletion of Durable Subscriptions 4
Setting Container Managed Transactions 4
Setting Message Filtering (JMS Selectors) 4
Controlling MDB Concurrency 4
Setting Subscription Identifiers 5
Setting Message Distribution Tuning 5
Setting topicMessagesDistributionMode 5
Setting distributedDestinationConnection 7
Best Practices 8
Warning about Non-Transactional MDBs in Compatibility Mode 8
Warning About Using Local RDTs with Durable MDBs 8
Warning About Using Local RDTs with Non-Durable MDBs 9
Warning about Changing Durable MDB Attributes, Topic Type, EJB Name 9
Choosing Between Partitioned and Replicated Topics 9
Choosing an MDB Topic Messages Distribution Mode 10
Managing and Viewing Subscriptions: 10
Handling Uneven Message Loads and/or Message Processing Delays 10
Configuring for Service Migration 10
Upgrading Applications from Previous Releases 11
Topic MDB Sample 11

11 Deployment Elements and Annotations for MDBs

A Topic Deployment Scenarios

How Configuration Permutations Determine Deployment Actions A-1
Typical Scenarios A-3
Standalone (Non-distributed) Topic Scenarios A-3
One-Copy-Per-Server A-3
One-Copy-Per-Application A-4
Replicated Distributed Topic Scenarios A-4
Scenario 1: Replicated DT, One Copy Per Server, Local Deployment, Local Only Consumption A-4

Copyright © 2007, 2026, Oracle and/or its affiliates.

Scenario 2: Replicated DT, One Copy Per Server, Local Deployment, Every Member Consumption A-5
Scenario 3: Replicated DT, One Copy Per Server, Remote Deployment A-7
Scenario 4: Replicated DT, One Copy Per Application, Local Deployment, Local Only Consumption A-9
Scenario 5: Replicated DT, One Copy Per Application, Local Deployment, Every Member Consumption A-10
Scenario 6: Replicated DT, One Copy Per Application, Remote Deployment A-11
Partitioned Distributed Topic Scenarios A-12
Scenario 7: Partitioned DT, One Copy Per Server, Local Deployment, Local Only Consumption A-12
Scenario 8: Partitioned DT, One Copy Per Server, Local Deployment, Every Member Consumption A-12
Scenario 9: Partitioned DT, One Copy Per Server, Remote Deployment A-13
Scenario 10: Partitioned DT, One Copy Per Application, Local Deployment, Local Only Consumption A-14
Scenario 11: Partitioned DT, One Copy Per Application, Local Deployment, Every Member Consumption A-15
Scenario 12: Partitioned DT, One Copy Per Application, Remote Deployment A-16

B Topic Subscription Identifiers

C How WebLogic MDBs Leverage WebLogic JMS Extensions

Copyright © 2007, 2026, Oracle and/or its affiliates.

Preface

This document explains how to design, develop, configure, and manage applications that include Oracle WebLogic Server 26 Message-Driven Beans (MDBs) using Jakarta Enterprise Edition (Jakarta EE).

Audience

This document is written for application developers who understand the Java programming language and Enterprise Java Bean (EJB) concepts and essentials.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customer access to and use of Oracle support services will be pursuant to the terms and conditions specified in their Oracle order for the applicable services.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

Related Resources

- *Tuning Performance of Oracle WebLogic Server*
- *Developing Jakarta Enterprise Beans for Oracle WebLogic Server*

Conventions

The following text conventions are used in this document:

Copyright © 2007, 2026, Oracle and/or its affiliates.

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for text entered by users.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that user enter.

Copyright © 2007, 2026, Oracle and/or its affiliates.

1

Understanding Message-driven Beans

A message-driven bean (MDB) is an enterprise bean that allows Jakarta EE applications to process messages asynchronously. An MDB acts as a JMS or JCA message listener. The messages are sent by the Jakarta EE components like application client, another enterprise bean, web components, or even a non-Jakarta EE application.
These are the key features of message-driven beans:

- Clients do not access message-driven beans through interfaces. A message-driven bean has only a bean class.
- A message-driven bean's instances retain no data or conversational state for a specific client. All instances of a message-driven bean are equivalent, allowing the EJB container to assign a message to any message-driven bean instance. The container can pool these instances to allow streams of messages to be processed concurrently.

When a message arrives, the container invokes the appropriate message listener method on the MDB. For a JMS MDB, this method is `onMessage`. The `onMessage` method may call helper methods, or it may invoke a session or entity bean to process the information in the message or to store it in a database.

For a JMS MDB using container-managed transactions with the `Required` transaction attribute, message receipt and all transactional work in `onMessage` that participates in the same transaction are committed or rolled back together. If the transaction is rolled back, the JMS provider's redelivery rules apply. A bean-managed transaction does not include message receipt. See Configuring Transaction Management Strategy for an MDB.

For information about design alternatives for message-driven beans, see MDBs and Messaging Models.

For a description of the overall EJB development process, see *Developing Enterprise Beans for Oracle WebLogic Server*.

JCA-Based MDBs

MDBs can receive messages through Jakarta Connectors resource adapters. Configure the `resource-adapter-jndi-name` element under `message-driven-descriptor` in `weblogic-ejb-jar.xml` to identify the resource adapter.
See the Jakarta Connectors 2.1 specification and resource-adapter-jndi-name in *Developing Enterprise Beans, Version 4.0, for Oracle WebLogic Server*.

Copyright © 2007, 2026, Oracle and/or its affiliates.

2

MDB Life Cycle

Examine the phases in the life cycle of an MDB instance and how you can configure an MDB to control the life cycle.
This chapter includes the following sections:

- Overview
- MDBs and Concurrent Processing
- Limitations for Multi-threaded Topic MDBs

Overview

An MDB implements loosely coupled or asynchronous business logic in which the response to a request need not be immediate. A message-driven bean receives messages from a JMS queue or topic, and performs business logic based on the message contents. It is an asynchronous interface between EJBs and JMS.
All instances of a message-driven bean are equivalent—the EJB container can assign a message to any MDB instance. The container can pool these instances to allow streams of messages to be processed concurrently.

The EJB container interacts directly with a message-driven bean—creating bean instances and passing JMS messages to those instances as necessary. The container initially populates each MDB free pool according to `initial-beans-in-free-pool`, which defaults to `0`. It creates additional instances on demand and may remove instances during operation.

MDBs and Concurrent Processing

MDBs support concurrent processing for both topics and queues. On an Oracle WebLogic Server instance, each MDB deployment maintains one or more MDB instance pools, also known as free pools, that hold MDB instances not currently servicing requests. The `max-beans-in-free-pool` setting, the Work Manager or execute-thread configuration, and the message-processing mode determine the maximum number of concurrent MDB invocations for each pool.
For more information about topics and queues, see MDBs and Messaging Models.

See Tuning Message-Driven Beans in *Tuning Performance of Oracle WebLogic Server*.

The number of free pools associated with an MDB deployment depends on the type of destination the MDB deployment is connected to. Typically, an MDB deployment is associated with a single free pool on each Oracle WebLogic Server instance that hosts the deployment. However, on each Oracle WebLogic Server instance that hosts the deployment, an MDB deployment connected to a WebLogic JMS distributed destination might have one free pool for each physical destination associated with the distributed destination. The number of free pools is automatically determined by the EJB container; and, for MDBs associated with a JMS destination, each MDB free pool always corresponds to a single JMS connection.

For queue-based MDBs, the container creates and manages the JMS sessions used to receive messages and obtains an available bean instance from the pool for each invocation.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Topic-based MDBs may process messages serially, dispatch messages received through one JMS session to multiple pooled bean instances, or use multiple JMS sessions for concurrent consumption. This is automatically determined by the MDB container based on the message processing pattern specified by MDB application settings, the type of topic, the work-manager, and the `max-beans-in-free-pool` setting. See Configuring and Deploying MDBs Using JMS Topics . Also see Tuning Message-Driven Beans in *Tuning Performance of Oracle WebLogic Server*.

Limitations for Multi-threaded Topic MDBs

Examine the limitations for multi-threaded topic MDBs.
Topic MDB concurrency depends on the JMS provider, transaction settings, topic distribution mode, and concurrency settings. The default behavior for non-transactional topic MDBs is to multi-thread the message processing. There are some limitations when using:

- Non-transactional topic MDBs that work with foreign (non-WebLogic) topics
- Non-transactional topic MDBs in `Compatibility` mode that consume from a WebLogic JMS topic and process messages with a WebLogic JMS Unit-of-Order (UOO) value

For details, see the **Caution** in Controlling MDB Concurrency.

Copyright © 2007, 2026, Oracle and/or its affiliates.

3

MDBs and Messaging Models

This chapter describes the point-to-point (queue) and publish/subscribe (topic) messaging models used by JMS-based MDBs in Oracle WebLogic Server.
These models are described in detail in *Developing JMS Applications for Oracle WebLogic Server*.

This chapter includes the following sections:

- Point-to-Point (Queue) Model
- Publish/Subscribe (Topic) Model
- Exactly-Once Processing

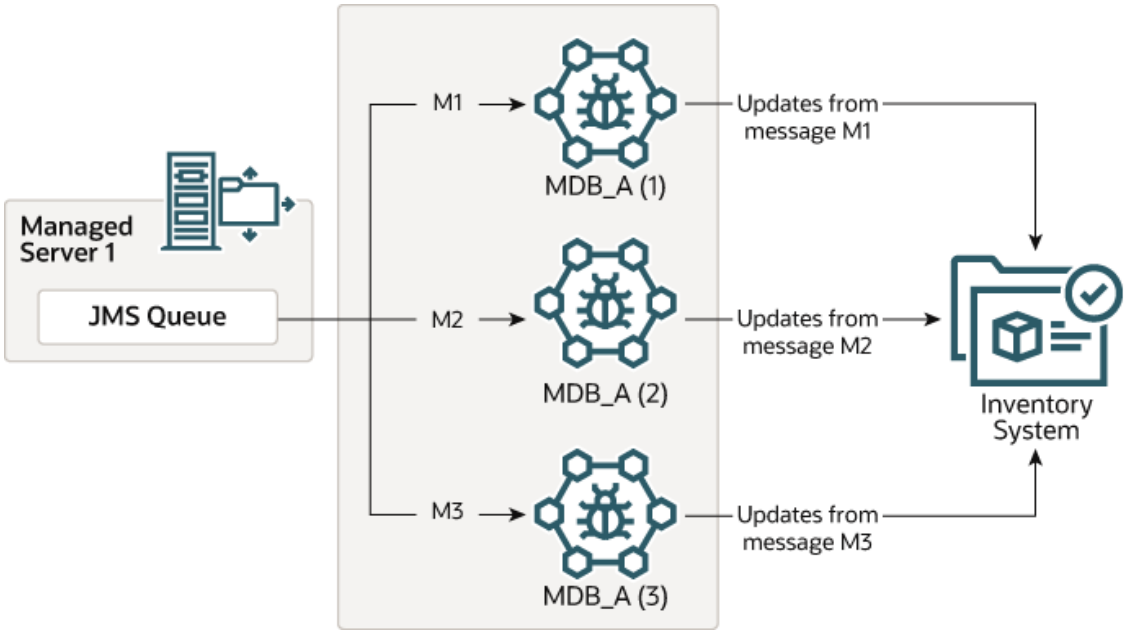
Point-to-Point (Queue) Model

In the point-to-point model, each message from a JMS queue is delivered to one MDB instance at a time. If a failure occurs before message acknowledgement or transaction commit, the message can be redelivered to an available MDB instance, subject to the JMS provider's redelivery and expiration settings. Redelivery does not require the same MDB instance to return.
Example: A department must update its back-end inventory system to reflect items sold throughout the day. Each message that decrements inventory must be processed once, and only once. It is not necessary for messages to be processed immediately upon generation or in any particular order, but it is critical that each message be processed.

Figure 3-1 illustrates a point-to-point application. Each message is processed by a single instance of MDB_A. Message "M1" is processed by MDB_A(1), "M2" is processed by MDB_A(2), and "M3" is processed by MDB_A(3).

Copyright © 2007, 2026, Oracle and/or its affiliates.

Figure 3-1 Point-to-Point Model



Publish/Subscribe (Topic) Model

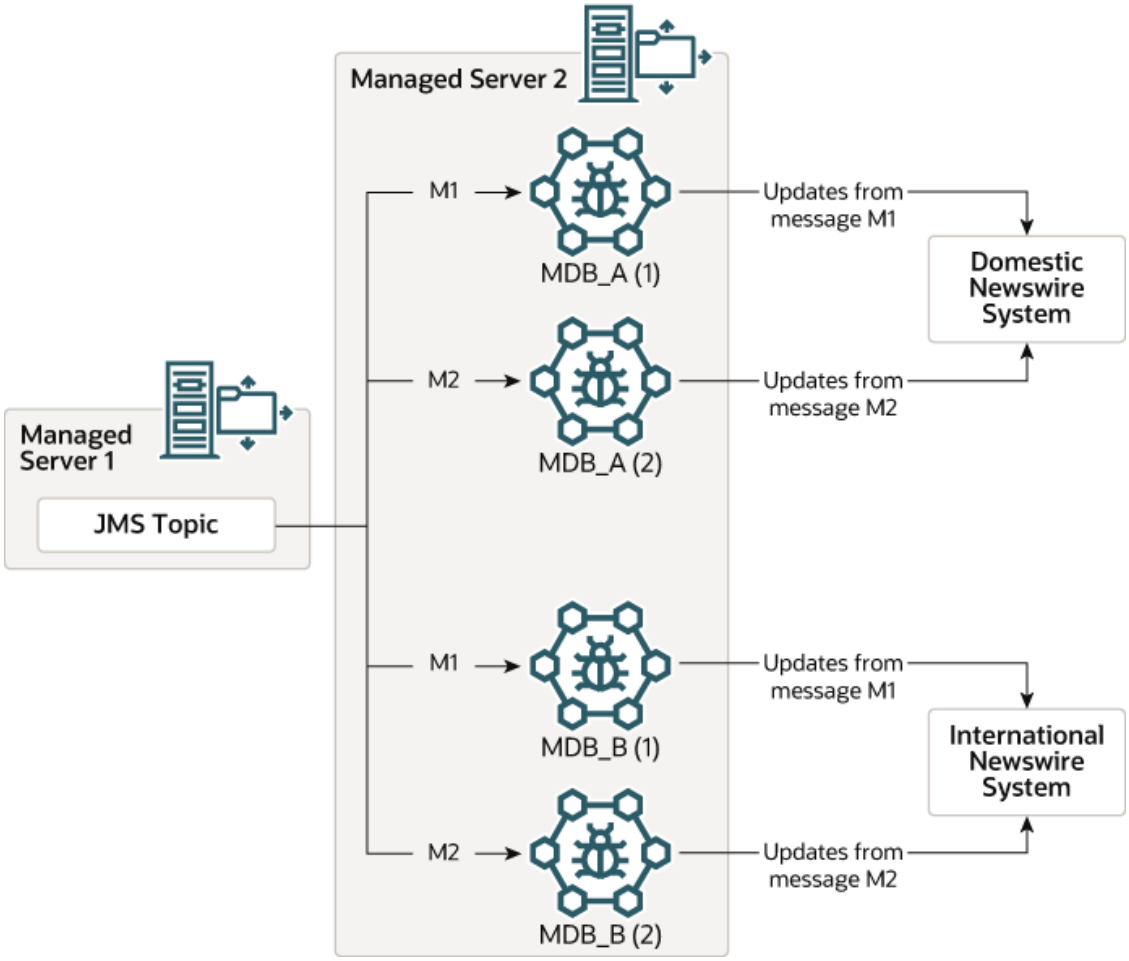
In the publish/subscribe model, a JMS topic publishes a copy of each message to each logical subscription. A logical subscription consists of one or more physical subscriptions, where each physical subscription is associated with a different member of a distributed topic.
For stand-alone (non-distributed) topics, a logical subscription always consists of a single physical subscription on the topic. If an MDB deployment goes down, it will miss messages published while it is unavailable unless it is configured to use a durable subscription. For information on durable subscriptions and for configuration instructions, see Setting Subscription Durability.

Example: A financial news service broadcasts stock prices and financial stories to subscribers, such as news wire services. Each message is distributed to each subscriber.

Figure 3-2 illustrates a publish/subscribe application. In contrast to a point-to-point application, in a publish/subscribe model, a copy of the message is processed for each of the logical subscriptions. In this diagram, there are two logical subscriptions, where each logical subscription consists of a separate physical subscription on the single topic. MDB_A has two instances that process the messages for a single dedicated subscription. Similarly, MDB_B has two instances that process the messages for a different single dedicated subscription. Message M1 is processed by an instance of MDB_A and an instance of MDB_B. Similarly, message M2 is processed by an instance of each of the subscribing MDBs.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Figure 3-2 Publish/Subscribe Model



Exactly-Once Processing

Use container-managed transactions with the `Required` transaction attribute to coordinate message receipt and participating transactional work. Redelivery can still cause repeated invocations of the MDB.
A JMS MDB can receive the same message more than once. To coordinate message receipt with transactional processing, use container-managed transactions with the `Required` transaction attribute. Message receipt and all transactional work in `onMessage()` that participates in the same transaction are committed or rolled back together. A rollback can lead to another invocation of `onMessage()`. Work outside that transaction is not rolled back and must tolerate repeated execution.

Redelivery is subject to the JMS provider's expiration and redelivery settings:

- If an application or server fails before message acknowledgement or transaction commit, or if the transaction rolls back, the message may be redelivered and processed again.
- Non-persistent messages can also be redelivered, but they can be lost if their JMS server shuts down or crashes.

See Configuring Transaction Management Strategy for an MDB and Redelivery and Exception Handling.

Copyright © 2007, 2026, Oracle and/or its affiliates.

4

Deploying MDBs

Examine the various approaches for deploying MDBs and the JMS destination to which the MDBs listen and understand how JMS distributes the messaging load across available members of the distributed destination.
This chapter includes the following sections:

- Destination and MDBs: Collocation vs. non-Collocation
- Collocated Destination/MDBs
- Non-Collocated Destination/MDBs
- JMS Distributed Destinations
- Best Practice

Destination and MDBs: Collocation vs. non-Collocation

You can deploy an MDB on the same server instance as the JMS destination on which it listens or on a separate server instance. When you deploy an MDB on the same server instance as the JMS destination on which it listens it is referred to as *collocation* and when you deploy an MDB on a separate server instance it is referred to as *non-collocation*.

Collocated Destination/MDBs

Collocating an MDB with the destination to which it listens keeps message traffic local and avoids network round trips. Collocation is the best choice if you use Oracle WebLogic Server JMS and want to minimize message processing latency and network traffic.
You cannot collocate the MDB and the JMS destination if you use a third-party JMS provider that cannot run on Oracle WebLogic Server, such as MQ Series.

Figure 4-1 and Figure 4-2 illustrate architectures in which the MDB application is deployed to the server instance that hosts the associated JMS destination. These architectures vary in that the first has a *distributed destination* and the second does not. In an application that uses distributed destinations, the MDB is deployed to each server instance that hosts a member of the distributed destination set. For more information about distributed destinations, see JMS Distributed Destinations. As illustrated in Figure 4-1 the message "M1" is delivered to an instance of MDB_A on each server instance where a distributed destination and MDB_A are deployed. Figure 4-1 illustrates a replicated distributed topic with one member on each server, using `topicMessagesDistributionMode=One-Copy-Per-Server` and `distributedDestinationConnection=LocalOnly`. Topic patterns are discussed in more detail in Setting topicMessagesDistributionMode.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Figure 4-1 Collocated Destination/MDBs, Replicated Distributed Topic, One-Copy-Per-Server Pattern

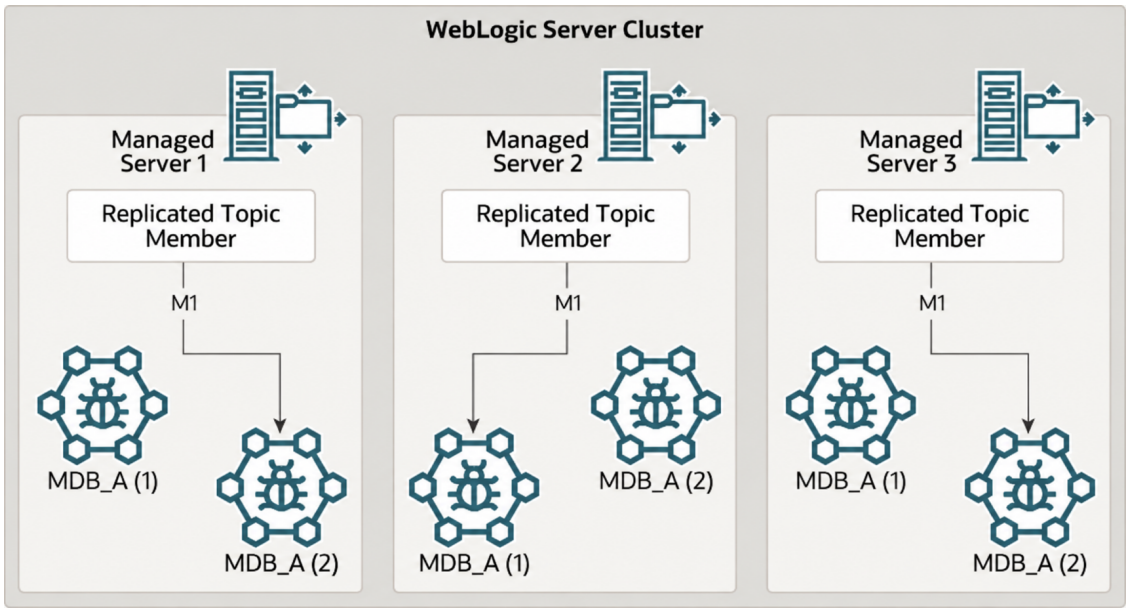
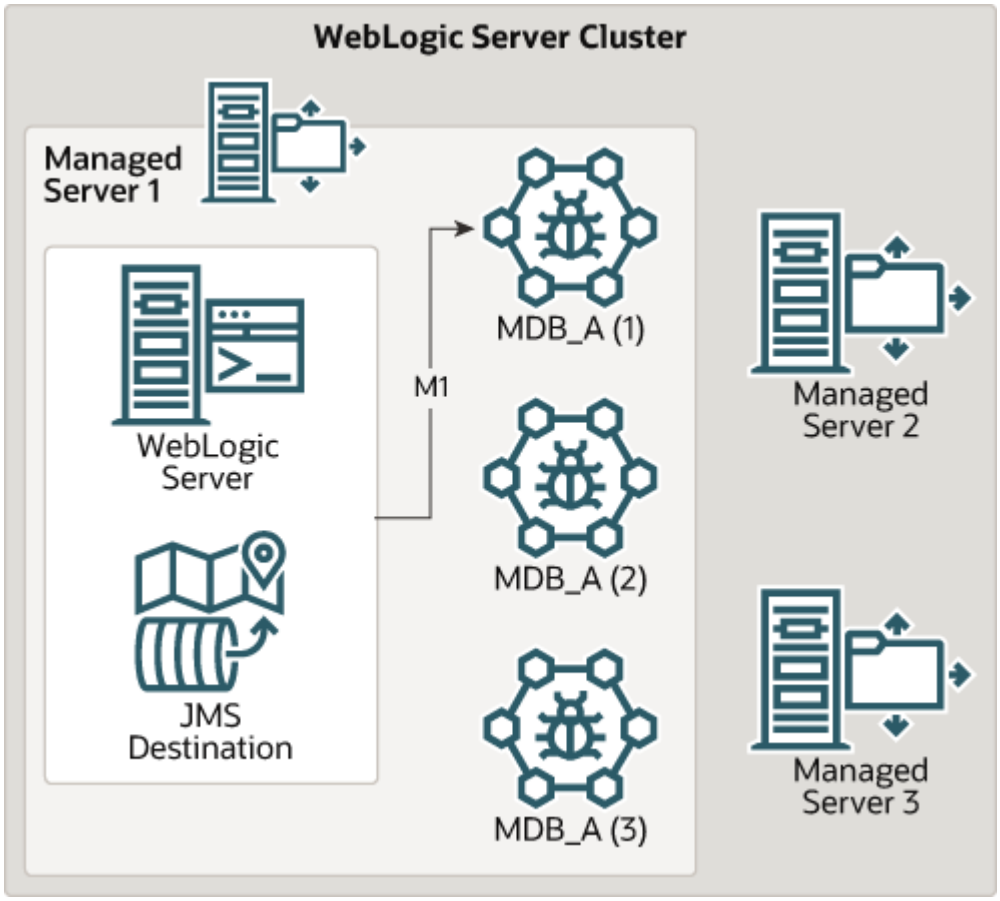


Figure 4-2 Collocated Destination/MDBs, Non-Distributed Destination

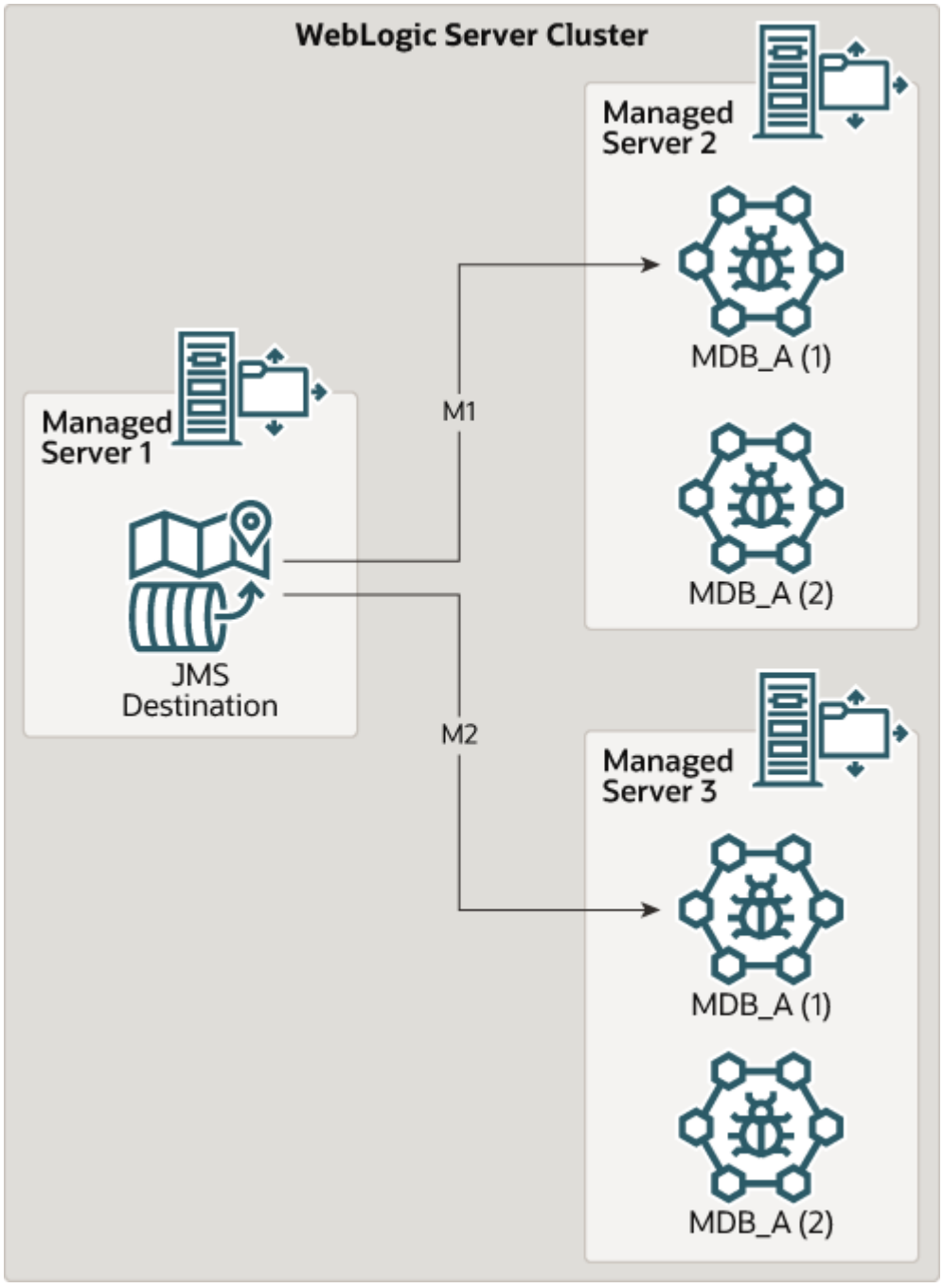


Copyright © 2007, 2026, Oracle and/or its affiliates.

Non-Collocated Destination/MDBs

Examine the non-collocated destination/MDBs in which an MDB runs on a different server instance from the JMS Destination. It is best suited for applications that use a third-party JMS provider or if you want to isolate application code (the MDBs) from the JMS infrastructure. Figure 4-3 illustrates an architecture in which an MDB runs on a different server instance from the JMS Destination to which the MDB listens.

Figure 4-3 Non-Collocated Application, Non-Distributed Destination



Copyright © 2007, 2026, Oracle and/or its affiliates.

Running your MDBs on a different server instance from the JMS Destination to which the MDB listens is suitable if:

- Your application uses a third-party JMS provider, such as MQ Series.
- You want to isolate application code (the MDBs) from the JMS infrastructure. By placing JMS destinations and MDBs on separate server instances, you prevent application problems—for example, MDBs consuming all of your virtual machine's memory—from interrupting the operation of the JMS infrastructure.
- Your MDB application is very CPU-intensive. On a separate machine, your application can use 100 percent of the CPU without affecting the operation of the JMS infrastructure.
- One machine in your configuration has more processing power, disk space, and memory than other machines in the configuration.

The JMS destination and MDBs could also be deployed on non-clustered servers, servers within the same cluster, or to servers in separate clusters.

> **Note**
>
> WebLogic Server does not support non-collocated Destination/MDBs in a cluster when you deploy the destination on a target that can be migrated.

JMS Distributed Destinations

When an MDB application runs on an Oracle WebLogic Server cluster, you can configure multiple physical destinations (queues or topics) as a *distributed destination*, which appears to message producers and consumers to be a single destination.
If you configure a distributed destination, WebLogic JMS distributes the messaging load across available members of the distributed destination. If a member of the destination becomes unavailable due to a server failure, message traffic is re-directed to the other available physical destinations in the distributed destination set. You control whether an MDB that accesses a WebLogic distributed queue in the same cluster consumes from all distributed destination members or only those members local to the current Oracle WebLogic Server, using the `distributed-destination-connection` element in the `weblogic-ejb-jar.xml` file or the `distributedDestinationConnection` activation configuration property. Similarly, this setting controls behavior for some topic MDB scenarios, as described in Configuring and Deploying MDBs Using JMS Topics , and Topic Deployment Scenarios.

If you deploy an MDB and the associated distributed queue to the same cluster, Oracle WebLogic Server automatically enumerates the distributed queue members and ensures that each member is serviced by at least one MDB pool. For distributed queues, there will be one MDB pool for each local member when `distributedDestinationConnection` is `LocalOnly` (the default); otherwise, for queues, when `distributedDestinationConnection` is set to `EveryMember`, each Oracle WebLogic Server instance creates multiple local MDB pools - one for each local member plus one for each remote member.

If you deploy an MDB and its associated queue to different clusters, Oracle WebLogic Server automatically enumerates the distributed queue members and ensures that each member is serviced by an MDB pool on each server in the MDB cluster. For example, if the distributed queue has three members, each JVM in the MDB cluster will create three MDB pools.

For more information about distributed topics, see Configuring and Deploying MDBs Using JMS Topics .

Copyright © 2007, 2026, Oracle and/or its affiliates.

Best Practice

WebLogic clustering and WebLogic JMS distributed destinations increase scalability and high availability. Examine some of the best practices to increase scalability and high availability.

Oracle recommends that the machines that host a cluster have identical or similar processing power, disk space, and memory to ensure well-load-balanced message processing. Similarly, it is recommended that the Oracle WebLogic Server instances in a particular WebLogic cluster have homogeneous JMS configuration and MDB deployments.

For an example, see Figure 4-1. For additional information about distributed destinations, see Using Distributed Destinations in *Developing JMS Applications for Oracle WebLogic Server*.

Copyright © 2007, 2026, Oracle and/or its affiliates.

5

Programming and Configuring MDBs: Main Steps

Examine the step-by-step instructions for implementing an MDB using pre-EJB 3.0 style XML descriptors to configure its behavior.
For a summary of key deployment elements for MDBs, see Deployment Elements and Annotations for MDBs. For an introduction to key deployment elements for topic MDBs, see Configuring and Deploying MDBs Using JMS Topics .

This chapter includes the following sections:

- Required JMS Configuration
- Create MDB Class and Configure Deployment Elements

Required JMS Configuration

Examine the steps to configure JMS connection factory and JMS destination.
The steps in the following section assume that you have access to the appropriate JMS components:

- A JMS connection factory.

  A connection factory must be XA transaction (global JTA transaction) capable in order to support transactional MDBs.

  The default WebLogic JMS MDB connection factory is XA-capable and is available on each WebLogic Server instance where default JMS connection factories are enabled. It is sufficient for the majority of MDBs that consume from WebLogic JMS destinations. For information about WebLogic JMS default connection factories, see Using Default Connection Factories Defined by WebLogic Server in *Administering JMS Resources for Oracle WebLogic Server*.

  For instructions about how to create a JMS system module see Create a JMS System Module in Oracle WebLogic Remote Console Online Help.

  The default behavior and configuration methods for other JMS provider connection factories vary. If you use a non-Oracle JMS provider, see the vendor documentation for details.

- A JMS destination

  For a summary of WebLogic Server messaging, see Messaging in Oracle WebLogic Remote Console Online Help.

Copyright © 2007, 2026, Oracle and/or its affiliates.

> **Note**
>
> If your JMS provider is a remote Oracle WebLogic Server JMS provider or a foreign JMS provider, and you use the wrapper approach recommended in Whether to Use Foreign JMS Server Mappings, in addition to configuring the non-local JMS components, you must also configure a Foreign Connection Factory and Foreign JMS Destination in your local JNDI tree.

Create MDB Class and Configure Deployment Elements

Examine the steps to implement a message-driven bean in an Oracle WebLogic Server. Use the following steps to implement a message-driven bean:

1. Create a source file (message-driven bean class) that implements both the `jakarta.ejb.MessageDrivenBean` and `jakarta.jms.MessageListener` interfaces.

   The MDB class must define the following methods:

   - One `ejbCreate()` method that the container invokes after creating each new instance of the MDB.
   - One `onMessage()` method that is called by the EJB container when a message is received. This method contains the business logic that processes the message.
   - One `setMessageDrivenContext()` method that provides information to the bean instance about its environment (certain deployment descriptor values); the MDB also uses the context to access container services. See Using the Message-Driven Bean Context.
   - One `ejbRemove()` method that the container invokes when removing the MDB instance, allowing the instance to release resources.

   > **Note**
   >
   > Most EJB 4.0 applications implement only `jakarta.jms.MessageListener`, which defines a single method - `onMessage()`. However, a message-driven bean is permitted to implement a listener interface with no methods. A bean that implements a no-methods interface, exposes all non-static public methods of the bean class and of any superclasses except `java.lang.Object` as message listener methods.

2. Declare the MDB in `ejb-jar.xml`, as illustrated in the following excerpt:

```
<ejb-jar>
    <enterprise-beans>
        <message-driven>
            <ejb-name>...</ejb-name>
            <ejb-class>...</ejb-class>
            <transaction-type>Container</transaction-type>
            <acknowledge-mode>Auto-acknowledge</acknowledge-mode>
            <message-driven-destination>
                <destination-type>jakarta.jms.Topic</destination-type>
                <subscription-durability>Durable</subscription-durability>
            </message-driven-destination>
        </message-driven>
    </enterprise-beans>
```


Copyright © 2007, 2026, Oracle and/or its affiliates.

```
    <assembly-descriptor>
        <container-transaction>
            <method>
                <ejb-name>...</ejb-name>
                <method-name>onMessage</method-name>
            </method>
            <trans-attribute>Required</trans-attribute>
        </container-transaction>
    </assembly-descriptor>
</ejb-jar>
```

The key behaviors to configure are:

- Transaction management strategy—The MDB's transaction management strategy, in the `transaction-type` element. For instructions, see Configuring Transaction Management Strategy for an MDB.
- Destination type—The type of destination to which the MDB listens. For more information, see Configuring Destination Type.

3. Configure WebLogic-specific behaviors for the MDB in the `message-driven-descriptor` element of `weblogic-ejb-jar.xml`. For example:

```
<weblogic-ejb-jar>
    <weblogic-enterprise-bean>
        <ejb-name>exampleMessageDrivenA</ejb-name>
        <message-driven-descriptor>
            <pool>...</pool>
            <timer-descriptor>...</timer-descriptor>
            <destination-jndi-name>...</destination-jndi-name>
            <initial-context-factory>...</initial-context-factory>
            <provider-url>...</provider-url>
            <connection-factory-jndi-name>...</connection-factory-jndi-name>
            <jms-polling-interval-seconds>...</jms-polling-interval-seconds>
            <jms-client-id>...</jms-client-id>
            <generate-unique-jms-client-id>...</generate-unique-jms-client-id>
            <durable-subscription-deletion>...</durable-subscription-deletion>
            <max-messages-in-transaction>...</max-messages-in-transaction>
            <init-suspend-seconds>...</init-suspend-seconds>
            <max-suspend-seconds>...</max-suspend-seconds>
        </message-driven-descriptor>
    </weblogic-enterprise-bean>
</weblogic-ejb-jar>
```

The key elements to configure are those that specify how to access the destination. In general, applications that follow best practices should never need to specify the `initial-context-factory` or `provider-url` fields. For instructions, see Configuring MDBs for Destinations.

4. Compile and generate the MDB class using the instructions in Compile Java Source in *Developing Enterprise Beans, Version 4.0, for Oracle WebLogic Server*.
5. Deploy the bean on Oracle WebLogic Server using the instructions in the section Preparing Applications and Modules for Deployment in *Deploying Applications to Oracle WebLogic Server*.

   If Oracle WebLogic Server cannot find an MDB's JMS destination during deployment, deployment succeeds, but Oracle WebLogic Server prints a message saying the destination was not found. The MDB container then periodically retries connecting to the JMS destination until it succeeds. For more information, see Migration and Recovery for Clustered MDBs.

Copyright © 2007, 2026, Oracle and/or its affiliates.

6

Programming and Configuring MDBs: Details

Examine the steps to program and configure MDBs.
The sections supplement the instructions in Programming and Configuring MDBs: Main Steps.

> **Note**
>
> This chapter uses a pre-EJB 3.0 deployment descriptor to illustrate basic MDB configuration. If you plan to use EJB 4.0 annotations, see also Deployment Elements and Annotations for MDBs and Using EJB 4.0 Compliant MDBs for the equivalent settings.

This chapter includes the following sections:

- Configuring Destination Type
- Configuring Transaction Management Strategy for an MDB
- Configuring MDBs for Destinations
- Configuring Message Handling Behaviors
- Using the Message-Driven Bean Context
- Configuring Suspension of Message Delivery During JMS Resource Outages
- Configuring the Number of Seconds to Suspend a JMS Connection
- Configuring a Security Identity for a Message-Driven Bean
- Using MDBs With Cross Domain Security
- Configuring EJBs to Use Logical Message Destinations

Configuring Destination Type

In an Enterprise Beans 4.0 `ejb-jar.xml` descriptor, set `message-destination-type` within the MDB's `message-driven` element. Alternatively, set the `destinationType` activation configuration property in `ejb-jar.xml` or in `@MessageDriven.activationConfig`.

- To specify a topic, set `message-destination-type` or the `destinationType` activation configuration property to `jakarta.jms.Topic`. For a topic, set the `subscriptionDurability` activation configuration property to `Durable` or `NonDurable` (the default). For important additional Topic related settings see Configuring and Deploying MDBs Using JMS Topics , and Deployment Elements and Annotations for MDBs.
- To specify a queue, set `message-destination-type` or the `destinationType` activation configuration property to `jakarta.jms.Queue`. For additional Queue related settings see Deployment Elements and Annotations for MDBs.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Configuring Transaction Management Strategy for an MDB

Learn how to configure an MDB for managing its own transactions or deferring transaction management to the container.
To configure container-level transaction management using descriptor elements:

- Set the `transaction-type` element in the `message-driven` element in the `ejb-jar.xml` file to `Container`.
- Set the `trans-attribute` element in the `container-transaction` element in `ejb-jar.xml` to `Required`.

> **Note**
>
> When `transaction-type` is `Container` and no transaction attribute is specified for `onMessage`, the default is `Required` for EJB 3.0 and later MDBs, including EJB 4.0 MDBs. For MDBs using a pre-EJB 3.0 deployment descriptor, WebLogic Server defaults to `NotSupported` and logs a warning. In that legacy case, explicitly specify `Required` to make message receipt and participating transactional work in `onMessage` part of the same container-managed transaction.

- To change the timeout period for the transaction, set `trans-timeout-seconds` in the `transaction-descriptor` element of `weblogic-ejb-jar.xml`. If a transaction times out, it is rolled back, and the message is redelivered. If `trans-timeout-seconds` is not specified or is set to `0`, the MDB uses the configured JTA transaction timeout, whose default is 30 seconds. For an application with long-running transactions, it may be appropriate to increase the timeout period.

To configure container-level transaction management using EJB annotations:

```
import jakarta.ejb.TransactionAttribute;
import jakarta.ejb.TransactionAttributeType;
...
@TransactionAttribute(value = TransactionAttributeType.REQUIRED)
public void onMessage(Message msg) {
...
```

To configure bean-level transaction management using descriptor elements:

- Set the `transaction-type` element in the `message-driven` element in the `ejb-jar.xml` file to `Bean`.
- Set the `acknowledge-mode` element to specify the desired JMS acknowledgment semantics, either one of the following:
  - `Auto-acknowledge` (the default), corresponding to `Session.AUTO_ACKNOWLEDGE`.
  - `Dups-ok-acknowledge`, corresponding to `Session.DUPS_OK_ACKNOWLEDGE`.

See Session in *Developing JMS Applications for Oracle WebLogic Server*.

Configuring MDBs for Destinations

Learn how to configure local and remote destination for the MDBs. Oracle WebLogic Server MDBs support WebLogic JMS destinations and foreign (non-Oracle) JMS provider destinations.

Copyright © 2007, 2026, Oracle and/or its affiliates.

For the configuration scenarios in this section, a local WebLogic JMS destination is hosted on the same WebLogic Server instance as the MDB, or on another server in the same cluster, and is accessible through the MDB's local JNDI tree. A WebLogic JMS destination on a server outside that cluster is remote, even if both servers run on the same machine. Servers in the same cluster can run on different machines. Use the remote-provider configuration described below when the destination requires access to a different JNDI context.

Destinations that run under a non-Oracle JMS provider are referred to as *foreign*. Foreign JMS providers have their own JNDI provider and foreign JMS objects do not share the same context with a Oracle WebLogic Server MDB—unless the foreign JMS objects are configured with mappings to appear in the MDB's JNDI context. This approach is discussed in Whether to Use Foreign JMS Server Mappings.

The nature of a destination—local versus remote and WebLogic JMS versus non-Oracle—governs the configuration alternatives available, and dictates to some extent how you configure these key elements in the `message-driven-descriptor` for the MDB in `weblogic-ejb-jar.xml`:

- `initial-context-factory`
- `provider-url`
- `destination-jndi-name`
- `connection-factory-jndi-name`

For foreign and remote destinations, the simplest configuration strategy is to use Oracle WebLogic Server foreign JMS server mappings. These mappings allow you to create a "symbolic link" between a JMS object in a third-party JNDI provider or in a different Oracle WebLogic Server cluster or domain, and an object in the local WebLogic JNDI tree.

To configure the elements in `message-driven-descriptor` for specific scenarios, see Common Destination Scenarios: Illustrations and Key Element Settings .

For more information on when foreign JMS server mappings are appropriate, and the rules for configuring the `message-driven-descriptor` in `weblogic-ejb-jar.xml`, see these sections:

Whether to Use Foreign JMS Server Mappings

Using mappings means configuring a Foreign Connection Factory and a Foreign Destination that correspond to remote JMS objects (either non-Oracle or WebLogic JMS) as entries in your local JNDI tree.

> **Note**
>
> The Foreign JMS Server mapping feature is not the same as the Foreign JNDI mapping feature. MDBs generally require that you use the Foreign JMS Server feature when mapping JMS resources into JNDI. The Foreign JNDI feature is meant for non-JMS resource types.

- The use of mappings is recommended if you use a foreign JMS provider or a remote WebLogic JMS provider. For more information on JMS mapping classes, see "Simplified Access to Remote or Foreign JMS Providers" in Enhanced Support for Using WebLogic JMS with EJBs and Servlets in *Developing JMS Applications for Oracle WebLogic Server*.
- If you use a mapping for either the connection factory or the destination, you must create and use mappings for each of these objects.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Whether or not you use mappings determines how you configure the `initial-context-factory` and `destination-jndi-name`.

How to Set provider-url

`provider-url` specifies the URL of the JNDI service used by the JMS provider for the destination to which the MDB listens.

- If the JMS provider is local to the MDB (by definition, WebLogic JMS), do not specify `provider-url`.
- If the JMS provider is remote, whether WebLogic JMS or a foreign provider, and:
  - You do not use mappings, specify `provider-url`.
  - You do use mappings, do not specify `provider-url`. The URL is implicitly encoded in the mapping.

How to Set initial-context-factory

`initial-context-factory` identifies the class that implements the initial context factory used by the JMS provider.

- If your JMS provider is WebLogic JMS, whether local or remote, do not specify `initial-context-factory`.
- If your JMS provider is foreign, and
  - you do not use mappings, specify the initial context factory used by the JMS provider.
  - you do use mappings, do not specify `initial-context-factory`.

How to Set destination-jndi-name

`destination-jndi-name` identifies the JNDI name of destination to which the MDB listens.

- If your JMS provider is local, specify the name bound in the local JNDI tree for the destination.
- If your JMS provider is remote WebLogic JMS or a foreign provider:
  - Without mappings, specify the destination's JNDI name in the remote provider's JNDI tree.
  - With mappings, specify the local JNDI name configured for the Foreign Destination that maps to the remote destination.

How to Set connection-factory-jndi-name

`connection-factory-jndi-name` identifies the JNDI name of the connection factory used by the JMS provider.

- If your JMS provider is local, do not specify `connection-factory-jndi-name` unless you have configured a custom connection factory that the MDB will use.

  Custom connection factories are used when the default Oracle WebLogic Server connection factory does not satisfy your application requirements. For example, you might configure a custom connection factory in order to specify a particular desired value for the `MessagesMaximum` attribute.

Copyright © 2007, 2026, Oracle and/or its affiliates.

The procedure for configuring resources for the module is described in Configure Resources for JMS System Modules in Oracle WebLogic Remote Console Online Help.

> **Note**
>
> If you configure a custom WebLogic JMS connection factory for an MDB, set its `AcknowledgePolicy` to `Previous`. For transactional message delivery, also enable `XAConnectionFactoryEnabled` in the connection factory's transaction parameters.

- If your JMS provider is remote or foreign:
  - Without mappings, specify the connection factory's JNDI name in the remote provider's JNDI tree. For remote WebLogic JMS, you may omit this setting when using the default MDB connection factory and that factory is enabled on the remote server.
  - With mappings, specify the local JNDI name configured for the Foreign Connection Factory that maps to the remote or foreign JMS provider's connection factory.

Common Destination Scenarios: Illustrations and Key Element Settings

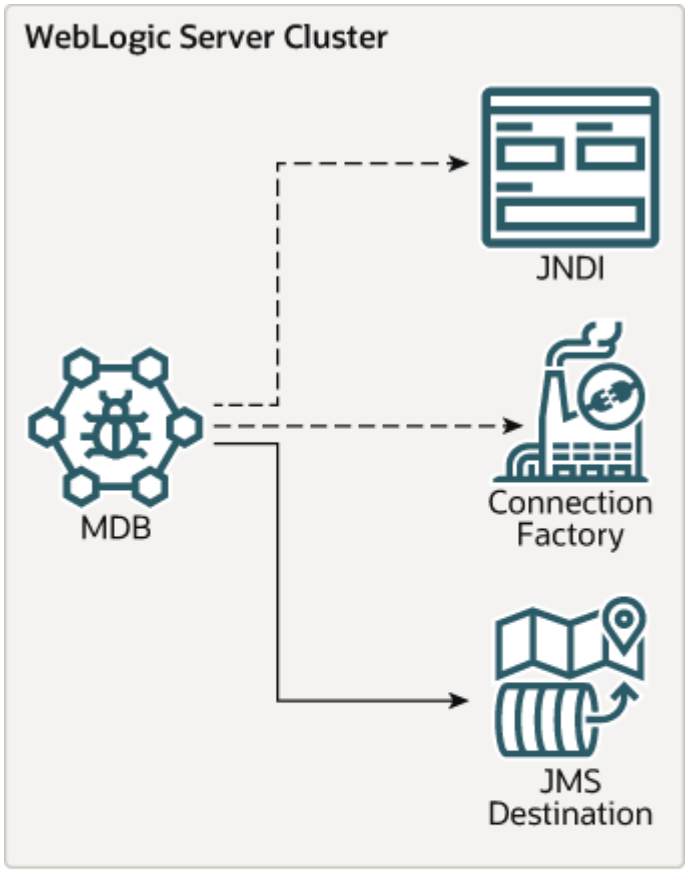
The figures in this section illustrate common destination configurations. For remote and foreign destinations, scenarios with and without mappings are included.

- Figure 6-1
- Figure 6-2
- Figure 6-3
- Figure 6-4

Table 6-1 summarizes how to configure the elements in the `message-driven-descriptor` element of `weblogic-ejb-jar.xml` for each scenario.

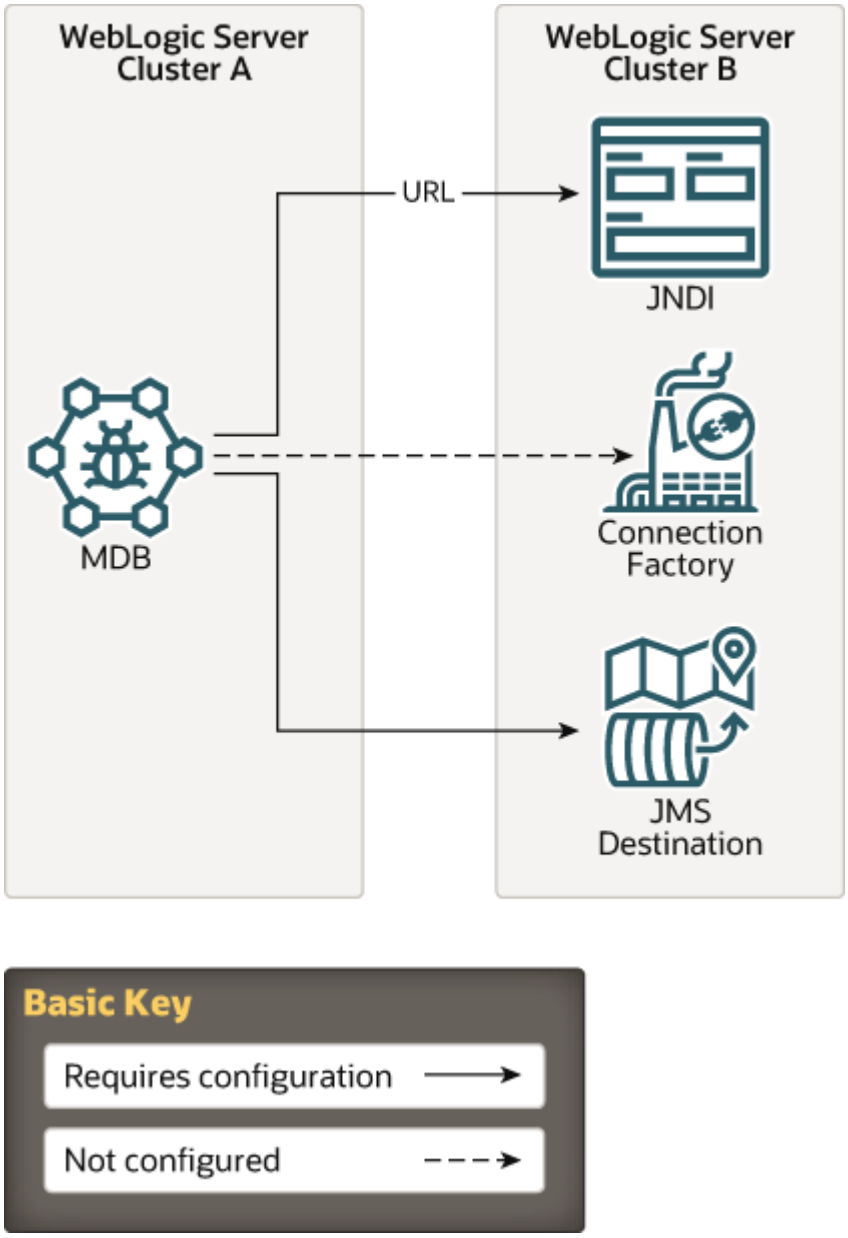
Copyright © 2007, 2026, Oracle and/or its affiliates.

Figure 6-1 A. Destination on a Local WebLogic JMS Server



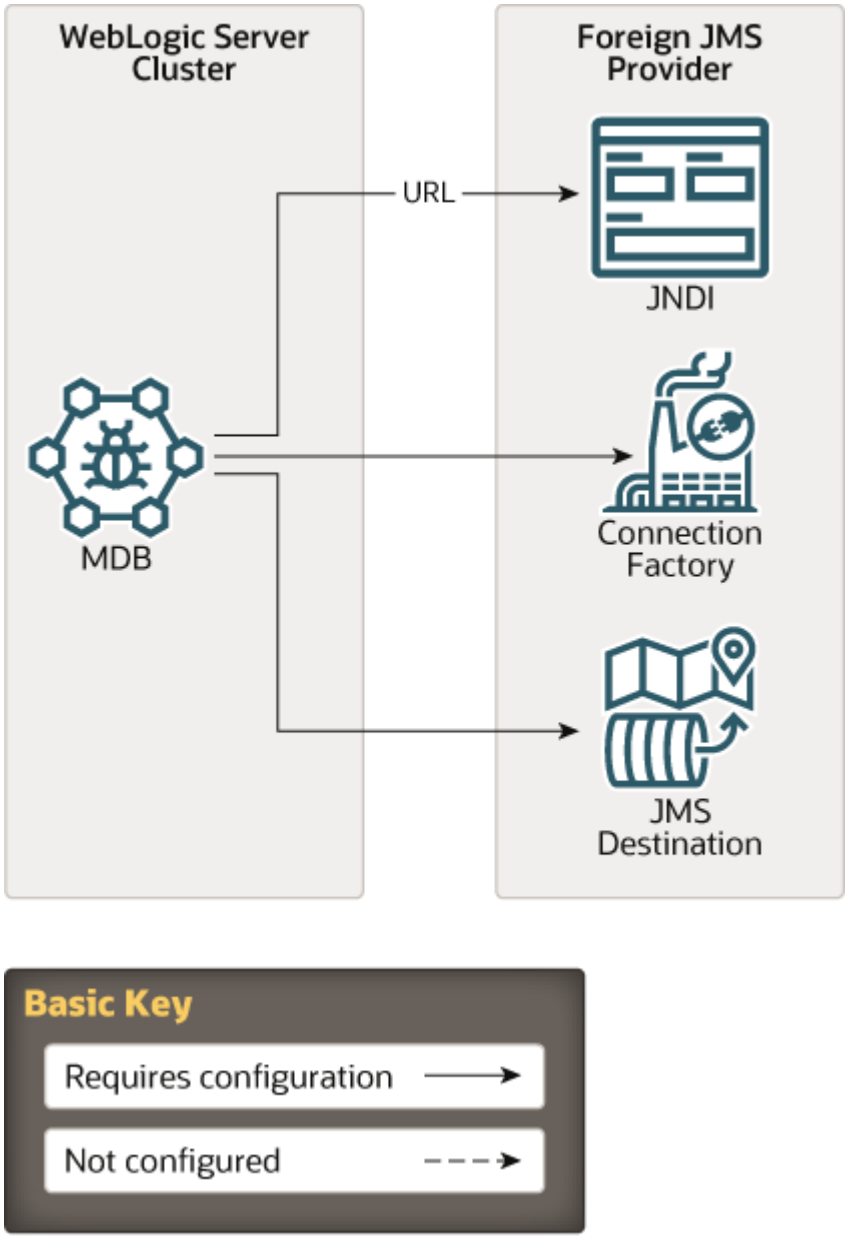
Copyright © 2007, 2026, Oracle and/or its affiliates.

Figure 6-2 B. Destination On a Remote WebLogic JMS Server—No Mappings



Copyright © 2007, 2026, Oracle and/or its affiliates.

Figure 6-3 C. Destination on Foreign JMS Server—No Mappings



Copyright © 2007, 2026, Oracle and/or its affiliates.

Figure 6-4 D. Destination on a Remote Oracle WebLogic Server or Foreign JMS Server —With Mappings

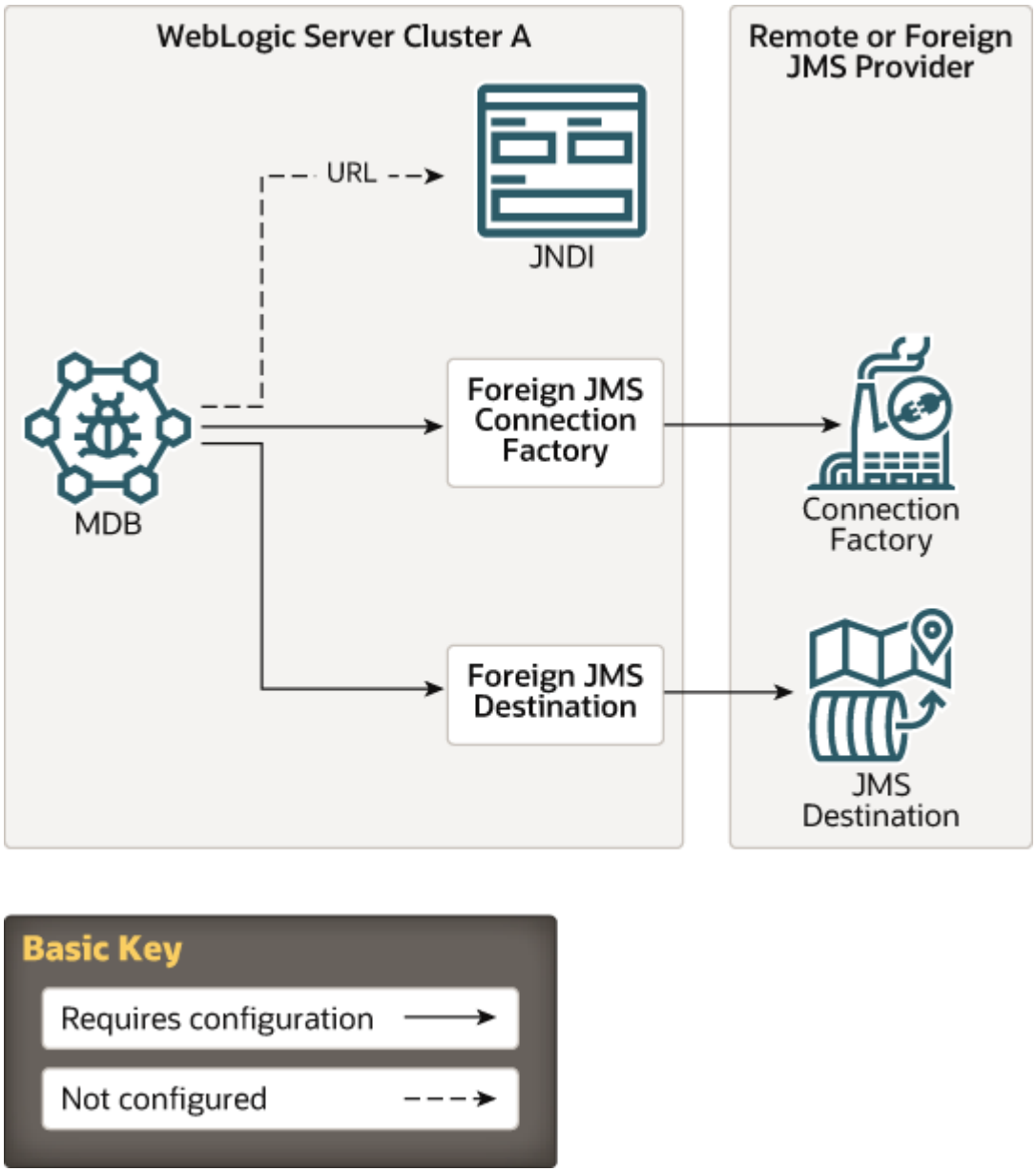


Table 6-1 Common Configuration Scenarios

Scenario	If destination is on...	Mappings Configured?	destination-jndi-name	initial-context-factory	provider-url	connection-factory-jndi-name
A	Local WebLogic JMS server	Not applicable for local WebLogic JMS server	Name of the local destination, as bound in local JNDI tree	Do not specify	Do not specify	Specify only if using a custom connection factory

Copyright © 2007, 2026, Oracle and/or its affiliates.

Table 6-1 (Cont.) Common Configuration Scenarios

Scenario	If destination is on...	Mappings Configured?	destination-jndi-name	initial-context-factory	provider-url	connection-factory-jndi-name
B	Remote WebLogic JMS Server	No mappings configured	Name of the remote destination, as bound in the remote JNDI tree	Do not specify	URL or cluster address for the remote WebLogic JMS Server	Specify only if using a custom connection factory on the remote provider
C	Foreign JMS Provider	No mappings configured	Name of the remote destination, as bound in the remote JNDI tree	Fully qualified class name of the foreign provider's initial context factory.	URL to access the foreign JMS provider	JNDI name of foreign connection factory
D	Remote Weblogic JMS Server or Foreign JMS server	Mappings configured	The name of the Foreign Destination—as bound in your local JNDI tree—that maps to the remote or foreign destination	Do not specify	Do not specify	The name of the Foreign Connection Factory—as bound in your local JNDI tree—that maps to the remote or foreign connection factory

Configuring Message Handling Behaviors

Examine the guidelines for behaviors related to message delivery:
This section includes the following topics:

- Ensuring Message Receipt Order
- Preventing and Handling Duplicate Messages
- Redelivery and Exception Handling

Ensuring Message Receipt Order

Make sure that the MDB's business logic allows for asynchronous message processing. Do not assume that MDBs receive messages in the order the client issues them.

When using WebLogic JMS destinations, Oracle recommends using the Unit-of-Order feature if strict ordering is required. This feature enforces ordering under all circumstances without requiring modification of the MDB, enables concurrent processing of sub-orderings that exist within the same destinations, and can be enabled via configuration or programmatically as appropriate. See Using Message Unit-of-Order in *Developing JMS Applications for Oracle WebLogic Server*.

If you are not using WebLogic destinations with unit-of-order to ensure that receipt order matches the order in which the client sent the message, you must do the following:

Copyright © 2007, 2026, Oracle and/or its affiliates.

- Set `max-beans-in-free-pool` to `1` for the MDB. This limits concurrent MDB instances to one per pool; it does not exclude other consumers of the destination.
- If your MDBs are deployed on a cluster, deploy them to a single node in the cluster.

Ensure that other consumers do not process the same ordered message stream concurrently. An MDB listening to a distributed destination can have a separate pool for each selected member, even when deployed on one server; a pool limit of one does not serialize processing across those members. Use Unit-of-Order when ordering is required across that configuration.

To ensure message ordering in the event of transaction rollback and recovery, configure a custom connection factory with `MessagesMaximum` set to 1, and ensure that no redelivery delay is configured. Foreign vendors have different names for the equivalent setting. This setting controls the number of messages that a vendor may push to a consumer before the consumer completes processing of its current message.

See Ordered Redelivery of Messages in *Developing JMS Applications for Oracle WebLogic Server*.

See `jakarta.jms.MessageListener.onMessage(Message)` in the Jakarta Messaging API documentation.

Preventing and Handling Duplicate Messages

The container handles acknowledgement of messages delivered to a JMS MDB. A failure before acknowledgement or transaction commit can cause a message to be redelivered, subject to the JMS provider's redelivery and expiration settings.

Your MDB design should allow for the likelihood of duplicate messages. Duplicate messages can be undesirable in certain cases. For example, if an MDB's `onMessage()` method includes code to debit a bank account, receiving and processing that message twice would result in the account being debited twice. In addition, re-sending messages consumes more processing resources.

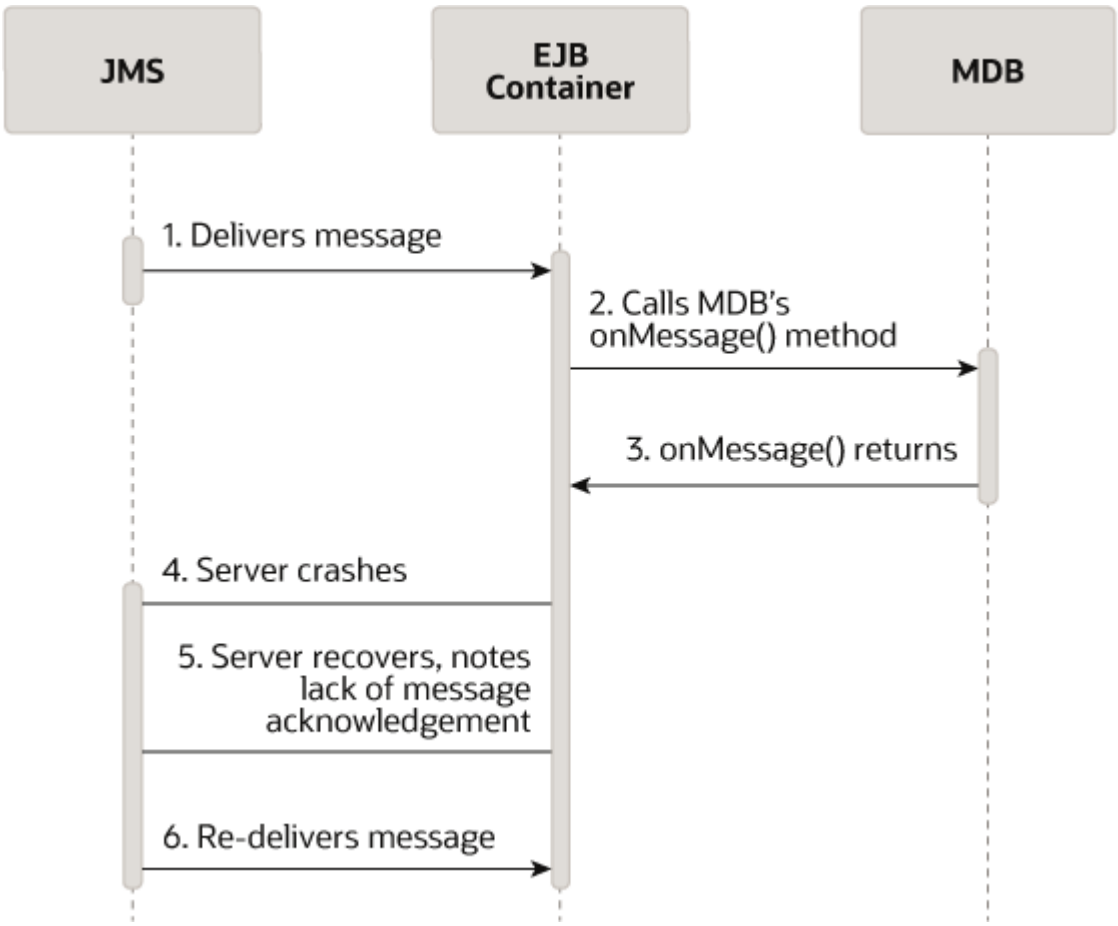
With container-managed transactions and the `Required` transaction attribute, message receipt and all transactional work in `onMessage` that participates in the same transaction are committed or rolled back together. A rollback can cause the message to be redelivered and `onMessage` to run again. This protects participating transactional work from duplicate committed effects caused by redelivery; it does not roll back nontransactional side effects. Transaction coordination can add CPU and disk overhead.

If the MDB manages its own transactions, your `onMessage()` code must handle duplicate messages, as receipt and acknowledgement occur outside of a transaction. In some applications, receipt and processing of duplicate messages is acceptable. In other cases, such as the bank account scenario, if a transaction is bean-managed, the bean code must prevent processing of duplicate messages. For example, the MDB could track messages that have been consumed in a database.

When message receipt is nontransactional, successful completion of `onMessage` does not by itself prevent redelivery. If the server crashes after the method returns but before the container acknowledges delivery, the message may be delivered again, as Figure 6-5 illustrates. This also applies when `onMessage` has already committed a bean-managed transaction, because that transaction does not include message receipt.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Figure 6-5 Server Crash Between Completion of onMessage() and Container Delivery Acknowledgement (Nontransactional Message Receipt)



Redelivery and Exception Handling

If an MDB is consuming a message when an unexpected error occurs, the MDB can throw a system exception that causes JMS to resend, delay, and then resend or give up, depending on how JMS is configured.

To force message redelivery for a transactional MDB, use the bean context to call `setRollbackOnly()`.

To force message redelivery for any MDB—transactional or non-transactional—you can throw an exception derived from the `RuntimeException` or `Error` thrown by the MDB. This causes the MDB instance to be destroyed and recreated, which incurs a performance penalty. If you want to avoid the overhead of a destroy and recreate, while still throwing a runtime exception, you can use a WebLogic extension. Throw an instance of a `weblogic.ejb.NonDestructiveRuntimeException`, for example,

```
throw new weblogic.ejb.NonDestructiveRuntimeException("force redelivery");
```

You may want to configure a redelivery delay based on what type of task the MDB's `onMessage()` method is performing. In some cases, redelivery should be instantaneous, for example, in an application that posts breaking news to a newswire service. In other cases, for example, if the MDB throws an exception because the database is down, redelivery should not occur immediately, but after the database is back up.

Copyright © 2007, 2026, Oracle and/or its affiliates.

> **Note**
>
> For fully ordered MDBs that do not use the Unit-of-Order feature, do not set a redelivery delay.

For instructions on configuring a redelivery delay, and other JMS exception handling features that can be used with MDB, see Managing Rolled Back, Recovered, Redelivered, or Expired Messages in *Developing JMS Applications for Oracle WebLogic Server*.

Using the Message-Driven Bean Context

Oracle WebLogic Server calls `setMessageDrivenContext()` to associate the MDB instance with a container context. Alternatively, EJB 4.0 MDB applications can specify an annotation that injects the MDB context. This is not a client context; the client context is not passed along with the JMS message.
To access the container context's properties from within the MDB instance, use the following methods from the `MessageDrivenContext` interface:

- `getCallerPrincipal()`—Returns the caller principal associated with the invocation. Do not assume that this is the principal of the JMS message producer.
- `isCallerInRole(String roleName)`—Tests the invocation's caller identity against the specified role. WebLogic Server permits this method for EJB 3.1 and later MDBs, including EJB 4.0 MDBs; it throws `IllegalStateException` for older descriptor versions.
- `setRollbackOnly()`—Can only be used by EJBs that use container-managed transactions.
- `getRollbackOnly()`— Can only be used by EJBs that use container-managed transactions.
- `getUserTransaction()`—Can only be used by EJBs that use bean-managed transaction demarcations.

  Both caller-identity methods may be used in message-listener methods, including `onMessage`, and their `AroundInvoke` interceptors, subject to the version requirement above. Neither method may be called during construction, dependency injection, or `PostConstruct` and `PreDestroy` callbacks. Timeout callbacks may use `getCallerPrincipal()`, but may not use `isCallerInRole()`.

> **Note**
>
> Although `getEJBHome()` is also inherited as part of the `MessageDrivenContext` interface, message-driven beans do not have a home interface. Calling `getEJBHome()` from within an MDB instance causes an `IllegalStateException`.

Configuring Suspension of Message Delivery During JMS Resource Outages

You can configure how an MDB behaves when the EJB container detects repeated failures during message processing, such as when an MDB throws the same exception ten times in succession.
You can configure:

Copyright © 2007, 2026, Oracle and/or its affiliates.

- An MDB to suspend the JMS connection and thereby stop receiving additional messages when the EJB container detects a JMS resource outage. If you choose this configuration option, you can specify:
  - The initial number of seconds the MDB should wait before it first resumes receiving messages.
  - The maximum number of seconds the MDB should wait before it resumes receiving messages.
- An MDB to not suspend the JMS connection when the EJB container detects a JMS resource outage.

When a JMS connection is suspended, message delivery is suspended for all JMS sessions associated with the connection. By default, when it detects a JMS resource outage, the EJB container suspends an MDB's JMS connection for `init-suspend-seconds`.

Configuring the Number of Seconds to Suspend a JMS Connection

You can suspend a JMS connection during a resource outage, which can be defined as an MDB throwing the same exception 10 times in succession.
To suspend an MDB's JMS connection, configure the following elements in the `weblogic-ejb-jar.xml` file:

- `init-suspend-seconds`—the initial amount of time, in seconds, to suspend a JMS connection when the EJB container detects a JMS resource outage. The default value is `5`.
- `max-suspend-seconds`—the maximum amount of time, in seconds, to suspend a JMS connection when the EJB container detects a JMS resource outage. The default value is `60`.

How the EJB Container Determines How Long to Suspend a JMS Connection

When automatic suspension is enabled (`max-suspend-seconds` is greater than `0`), repeated delivery failures cause the container to suspend the JMS connection for `init-suspend-seconds`. If `init-suspend-seconds` is `0`, the container uses an initial delay of 5 seconds. When the delay expires, the container attempts to resume delivery.

If repeated failures trigger another suspension, the next delay is twice the preceding delay, capped at `max-suspend-seconds`. With the defaults, successive delays are 5, 10, 20, 40, and 60 seconds, with subsequent delays remaining at 60 seconds while failures continue. Successful message processing resets the failure count and initial delay.

Setting `max-suspend-seconds` to `0` disables this automatic suspension, as described in Turning Off Suspension of a JMS Connection.

Turning Off Suspension of a JMS Connection

If you do not want an MDB's JMS connection to be suspended when the EJB container detects a resource outage, set the value of `max-suspend-seconds` to `0`. When the value of `max-suspend-seconds` is `0`, the value of `init-suspend-seconds` is ignored.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Configuring a Security Identity for a Message-Driven Bean

An MDB's configured run-as identity determines the security identity under which its message-processing method executes. The security identity used to establish a JMS connection is determined separately, based on the JMS provider and the configured authentication mechanism. When EJB component credential mapping is used, the container can obtain the credentials required to connect to the JMS provider.
To configure a security identity for an MDB:

1. Create a WebLogic user to serve as the MDB's local run-as identity. This user does not need to have the same username or password as the account used to authenticate to a remote JMS provider.
2. In `ejb-jar.xml`, declare the MDB's run-as role within its `message-driven` element:

   ```
   <security-identity>
     <run-as>
       <role-name>jmsrole</role-name>
     </run-as>
   </security-identity>
   ```

3. In the `ejb-jar.xml` deployment descriptor, declare the corresponding security role within the `assembly-descriptor` element:

   ```
   <assembly-descriptor>
     <security-role>
       <role-name>jmsrole</role-name>
     </security-role>
   </assembly-descriptor>
   ```

4. In `weblogic-ejb-jar.xml`, add the following `security-role-assignment` under the root `weblogic-ejb-jar` element to map `jmsrole` to the WebLogic user created in Step 1:

   ```
   <security-role-assignment>
     <role-name>jmsrole</role-name>
     <principal-name>username</principal-name>
   </security-role-assignment>
   ```

   Here, `username` is the name of the WebLogic user created in Step 1.
5. If the MDB uses EJB component credential mapping to authenticate to the JMS provider, configure a mapping from the MDB's local run-as identity to the username and password required by that provider. Other supported authentication mechanisms, such as credentials configured for a Foreign JMS Server, may also be used.

For MDBs accessing WebLogic JMS in another domain, see Using MDBs With Cross Domain Security for additional security configuration requirements.

Using MDBs With Cross Domain Security

Learn how to configure cross domain security for MDBs.
Cross Domain Security secures internal communication between WebLogic domains.
Configure the MDB's JNDI and JMS authentication and destination permissions separately.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Successful JMS connection authentication and message consumption do not establish that destination membership notifications or transaction coordination can complete successfully.

When an MDB consumes from a WebLogic JMS destination in another domain, apply the following requirements:

- **Standalone destination, without a transaction spanning WebLogic domains:** Message receipt alone does not require Cross Domain Security for distributed-destination membership or inter-domain transaction coordination. Other operations performed by the MDB may introduce additional requirements.
- **Distributed destination:** Configure Cross Domain Security between the MDB domain and the destination domain for internal communication used by the JMS Configuration Directory Service (CDS) to report destination membership and availability changes. This requirement also applies when message receipt is nontransactional.
- **Transactions spanning WebLogic domains, for either destination type:** Configure compatible transaction communication using Cross Domain Security or Security Interoperability Mode, as described in *Configuring Secure Inter-Domain and Intra-Domain Transaction Communication* in *Developing JTA Applications for Oracle WebLogic Server*. Security Interoperability Mode applies to JTA communication; it does not replace the distributed-destination requirement above.

Keep the participating domains' security configurations consistent in both directions, including communication back to the MDB domain. When using Cross Domain Security, configure cross-domain users and credential mappings for each communicating domain pair. For interoperability with domains that do not use Cross Domain Security, follow the excluded-domain guidance in *Configuring Cross-Domain Security* in *Administering Security for Oracle WebLogic Server*. General symmetry advice does not waive the distributed-destination requirement above.

Configuring EJBs to Use Logical Message Destinations

You can declare logical message destinations in an EJB deployment descriptor and map them to JMS queues or topics. Enterprise bean code can access a mapped destination through a message destination reference, and the container can use a logical destination link to select the source of messages for an MDB.

> **Note**
>
> Logical destinations and application-scoped destinations are not commonly used and are for advanced users only. For most users, Oracle recommends using the methods discussed in Configuring MDBs for Destinations.

A logical destination name identifies a deployment-time declaration. Bean code looks up the reference's environment name, such as `java:comp/env/jms/ExpenseQueueRef`. To configure an MDB listener, use `message-destination-link` directly under that bean's `message-driven` element.

For information on how unresolved and unavailable message destinations are handled, see EJBs and Message Destination References in *Developing Enterprise Beans, Version 4.0, for Oracle WebLogic Server*.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Declaring and Mapping a Logical JMS Destination

The following three examples assume that a JMS queue is already configured in a JMS system module with the JNDI name `jms/ExpenseQueue` and is accessible from the MDB's local JNDI tree. For existing applications that use packaged JMS modules, see **Using Application-Scoped JMS Resources** below.

In `ejb-jar.xml`, add this declaration inside `assembly-descriptor`:

```
<message-destination>
  <message-destination-name>ExpenseProcessingQueue</message-destination-name>
</message-destination>
```

In `weblogic-ejb-jar.xml`, add this mapping directly under `weblogic-ejb-jar`:

```
<message-destination-descriptor>
  <message-destination-name>ExpenseProcessingQueue</message-destination-name>
  <destination-jndi-name>jms/ExpenseQueue</destination-jndi-name>
</message-destination-descriptor>
```

The logical name must match in the declaration and mapping. This mapping refers to an existing destination; it does not create a JMS queue or define a JMS module.

Configuring a Message Destination Reference for Enterprise Bean Code

To let an enterprise bean send messages to the mapped queue, add this reference inside that bean's declaration in `ejb-jar.xml`:

```
<message-destination-ref>
  <message-destination-ref-name>jms/ExpenseQueueRef</message-destination-ref-
name>
  <message-destination-type>jakarta.jms.Queue</message-destination-type>
  <message-destination-usage>Produces</message-destination-usage>
  <message-destination-link>ExpenseProcessingQueue</message-destination-link>
</message-destination-ref>
```

The bean can look up `java:comp/env/jms/ExpenseQueueRef` to obtain the queue. This reference makes the destination available to bean code; it does not configure the MDB listener's incoming destination.

Linking an MDB to a Logical JMS Destination

Use the logical destination link and its mapping to configure the MDB's destination. Avoid conflicting destination configurations in annotations, deployment descriptors or deployment plans.

To have the container deliver messages from the mapped queue to an MDB, add these elements inside that MDB's `message-driven` declaration in `ejb-jar.xml`:

```
<message-destination-type>jakarta.jms.Queue</message-destination-type>
<message-destination-link>ExpenseProcessingQueue</message-destination-link>
```


Copyright © 2007, 2026, Oracle and/or its affiliates.

`message-destination-type` specifies the expected destination type. If a `destinationType` activation property is also configured, use the same value.

Place these elements after `transaction-type` and before `activation-config`, if those elements are present. Do not add `message-destination-usage` directly under `message-driven`; that element belongs to `message-destination-ref`.

Each MDB that uses this logical destination must declare its own link. A logical destination does not automatically configure every MDB in the application. To refer to a logical destination declared in another module, use the supported relative-module-path form, such as `other-ejb.jar#ExpenseProcessingQueue`, and provide the declaration and mapping in that module.

Using Application-Scoped JMS Resources

WebLogic JMS application modules, including packaged and standalone modules, are deprecated. For new applications, configure JMS resources in JMS system modules. This subsection explains resource links for existing applications that use packaged JMS modules.

In these applications, JMS modules are packaged within an enterprise application (EAR) and declared in `weblogic-application.xml`. A JMS module defines resources such as queues, topics and connection factories. These resources can be configured for use only by the enclosing application. Packaging a module and configuring its resources establishes their deployment and visibility; declaring a logical destination and its references establishes how enterprise beans refer to those resources.

Each MDB must still be configured to consume from its intended destination. Application scope does not automatically associate all MDBs with one destination. Different MDBs in the same application can use different resources.

For module configuration, resource scope, targeting and deployment guidance, see Configuring JMS Application Modules for Deployment (Deprecated) in *Administering JMS Resources for Oracle WebLogic Server*. The following descriptor fragments illustrate the names that must match in a packaged application.

For example, the EAR's `META-INF/weblogic-application.xml` contains this JMS module declaration under `weblogic-application`:

```
<module>
  <name>ExpenseJMS</name>
  <type>JMS</type>
  <path>jms/expense-resources-jms.xml</path>
</module>
```

The file at that path, relative to the EAR root, defines the resource:

```
<weblogic-jms xmlns="http://xmlns.oracle.com/weblogic/weblogic-jms">
  <queue name="ExpenseQueue">
    <sub-deployment-name>ExpenseQueueTarget</sub-deployment-name>
  </queue>
</weblogic-jms>
```

The deployment configuration must map the `ExpenseQueueTarget` subdeployment within the `ExpenseJMS` module to the JMS server that hosts the queue. The queue in this example omits both `jndi-name` and `local-jndi-name` and is accessed through its application resource link.

Copyright © 2007, 2026, Oracle and/or its affiliates.

To map the earlier logical destination to that queue, replace the earlier `message-destination-descriptor` mapping in `weblogic-ejb-jar.xml` with:

```
<message-destination-descriptor>
  <message-destination-name>ExpenseProcessingQueue</message-destination-name>
  <destination-resource-link>ExpenseJMS#ExpenseQueue</destination-resource-
link>
</message-destination-descriptor>
```

Keep this mapping directly under `weblogic-ejb-jar`. The resource-link format is `JMSModuleName#ResourceName`:

- `JMSModuleName` must match `<module><name>` in `weblogic-application.xml`, including capitalization: `ExpenseJMS` in this example. The `<path>` identifies the JMS descriptor file and is not used in the resource link.
- `ResourceName` must match the destination's `name` attribute in that JMS descriptor: `ExpenseQueue` in this example. It is separate from the JNDI name, the subdeployment name `ExpenseQueueTarget`, and the logical destination name `ExpenseProcessingQueue`.

This mapping refers to the module and queue configured above; it does not create them. A `message-destination-link` such as `other-ejb.jar#ExpenseProcessingQueue` uses a different naming convention to locate a logical declaration in another EJB module.

Retain the earlier logical `message-destination` declaration, any bean-code references that are needed, and each consuming MDB's `message-destination-type` and `message-destination-link`. The logical link remains `ExpenseProcessingQueue`; only the WebLogic mapping changes to identify the application-scoped queue.

Copyright © 2007, 2026, Oracle and/or its affiliates.

7

Using EJB 4.0 Compliant MDBs

Examine how to program and implement EJB 4.0 compliant MDBs:
This chapter includes the following sections:

- Implementing EJB 4.0 Compliant MDBs
- Programming EJB 4.0 Compliant MDBs

Implementing EJB 4.0 Compliant MDBs

You can implement MDBs that comply with EJB 4.0.
To implement EJB 4.0 compliant MDBs, follow the steps described in Overview of the EJB Development Process in *Developing Jakarta Enterprise Beans for Oracle WebLogic Server*.

Programming EJB 4.0 Compliant MDBs

You can program MDBs that comply with EJB 4.0.
To program EJB 4.0 compliant MDBs, follow the steps described in Programming the Bean File: Typical Steps in *Developing Jakarta Enterprise Beans for Oracle WebLogic Server*.

To declare an MDB with annotations, annotate its bean class with `@jakarta.ejb.MessageDriven`. Alternatively, declare the MDB using the `message-driven` element in `ejb-jar.xml`. The annotation provides the following optional attributes:

- `messageListenerInterface`—Specifies the message listener interface, if you have not explicitly implemented it or if the bean implements additional interfaces.

  > **Note**
  >
  > The bean class must implement, directly or indirectly, the message listener interface required by the messaging type that it supports or the methods of the message listener interface. In the case of JMS, this is the `jakarta.jms.MessageListener` interface.

- `activationConfig`—Specifies an array of activation configuration properties that configure the bean in its operational environment.

  Activation configuration properties are name-value pairs used to configure an MDB. Declare them in the `activation-config` element of `ejb-jar.xml`, or as `@ActivationConfigProperty` entries in the `activationConfig` attribute of `@MessageDriven`. If an activation configuration property is specified in both the annotation and the deployment descriptor, the deployment descriptor value takes precedence. An example using the `@ActivationConfigProperty` annotation is shown in Example 7-1. An example using the `ejb-jar.xml` deployment descriptor is shown in Example 7-2.

  To set activation configuration properties in `ejb-jar.xml`, add an `activation-config` element within the MDB's `message-driven` declaration, and put each `activation-config-property` inside it, as shown in Example 7-2.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Because activation configuration properties can be set in an `ejb-jar.xml` deployment descriptor or by using `@ActivationConfigProperty` annotation properties, conflicts may result if the same name is used in both places. Conflicts may also result from using same-named properties from pre-3.0 versions of EJB or from proprietary Oracle WebLogic Server EJB annotations. For overlapping settings, WebLogic Server applies the following precedence, from highest to lowest:

1. Properties set in the `weblogic-ejb-jar.xml` deployment descriptor
2. Proprietary Oracle WebLogic Server 10.0 (and later) annotations
3. `activation-config-property` properties set in the `ejb-jar.xml` deployment descriptor
4. `@ActivationConfigProperty` annotation properties

For example, if the same property exists in the `weblogic-ejb-jar.xml` descriptor and the `ejb-jar.xml` descriptor, the one in `weblogic-ejb-jar.xml` has the higher priority and overrides the value in `ejb-jar.xml`. Or, if the same property is set in both an `ejb-jar.xml` descriptor element and in an `@ActivationConfigProperty` annotation, the descriptor element takes precedence and the annotation is ignored.

For more information about activation configuration properties, see jakarta.ejb.ActivationConfigProperty in the *Jakarta EE Platform API*. Also see Table 11-1, which summarizes the activation configuration properties supported by Oracle WebLogic Server.

> **Note**
>
> Based on the Enterprise Bean specification, the `jakarta.ejb.ActivationConfigProperty` annotation is used for MDBs only. This annotation is not used for session or entity beans.

For detailed information on developing MDBs to support the messaging modes as described in MDBs and Messaging Models, see Programming and Configuring MDBs: Details.

Example 7-1 Example @ActivationConfigProperty Code

Apply the following annotation to the MDB class. This example uses the WebLogic-specific `connectionFactoryJndiName` and `initialContextFactory` activation properties for a JMS MDB. Configure the MDB's destination as described in Configuring MDBs for Destinations.

```
import jakarta.ejb.ActivationConfigProperty;
import jakarta.ejb.MessageDriven;

@MessageDriven(activationConfig = {
    @ActivationConfigProperty(
        propertyName = "connectionFactoryJndiName",
        propertyValue = "JNDINameOfMDBSourceCF"),
    @ActivationConfigProperty(
        propertyName = "initialContextFactory",
        propertyValue = "weblogic.jndi.WLInitialContextFactory")
})
```

Example 7-2 Activation Configuration Properties Set in ejb-jar.xml

Insert the following fragment in the MDB's existing `message-driven` declaration. The queue must already be available under the JNDI name `myQueue`.

Copyright © 2007, 2026, Oracle and/or its affiliates.

```
<activation-config>
  <activation-config-property>
 <activation-config-property-name>destinationLookup</activation-config-property-name>
 <activation-config-property-value>myQueue</activation-config-property-value>
  </activation-config-property>
  <activation-config-property>
 <activation-config-property-name>destinationType</activation-config-property-name>
 <activation-config-property-value>jakarta.jms.Queue</activation-config-property-value>
  </activation-config-property>
</activation-config>
```

MDB Sample Using Annotations

Example 7-3 shows an MDB that receives messages from a WebLogic JMS queue, processes them in a container-managed transaction, and forwards them to a target destination.

The MDB connects using JMS connection factory `MyCF` to receive from queue `MyQueue`. It forwards the messages to `MyTargetDest` using a connection generated from connection factory `MyTargetCF`.

This example assumes that `MyQueue`, `MyCF`, `MyTargetCF` and `MyTargetDest` are configured and available in the MDB's local JNDI tree. For WebLogic JMS connection factories defined in a JMS module, enable `XAConnectionFactoryEnabled` on both `MyCF` and `MyTargetCF` to support transactional receipt and forwarding. For more about MDB connection-factory configuration, see How to Set connection-factory-jndi-name.

The `Required` transaction attribute causes `onMessage` to execute in a container-managed transaction. Message receipt and JMS operations that participate in that transaction are committed or rolled back together. The call to `createSession(false, Session.AUTO_ACKNOWLEDGE)` does not, by itself, exclude a WebLogic-managed JMS session from participating in the container transaction.

Resource reference pooling note: The MDB uses a resource reference to access `MyTargetCF`. The resource reference automatically enables JMS producer pooling, as described in Enhanced Support for Using WebLogic JMS with EJBs and Servlets in *Developing JMS Applications for Oracle WebLogic Server*.

For a similar sample using topics instead of queues, see Example 10-1.

Example 7-3 Sample MDB Using Distributed Queues

```
package test;
import jakarta.annotation.Resources;
import jakarta.annotation.Resource;
import jakarta.ejb.ActivationConfigProperty;
import jakarta.ejb.MessageDriven;
import jakarta.ejb.MessageDrivenContext;
import jakarta.ejb.TransactionAttribute;
import jakarta.ejb.TransactionAttributeType;
import jakarta.jms.*;

@MessageDriven(
  name = "MyMDB",
  activationConfig = {
    @ActivationConfigProperty(propertyName  = "destinationType",
                              propertyValue = "jakarta.jms.Queue"),

    @ActivationConfigProperty(propertyName  = "connectionFactoryJndiName",
                              propertyValue = "MyCF"), // External JNDI Name

    @ActivationConfigProperty(propertyName  = "destinationJndiName",
```


Copyright © 2007, 2026, Oracle and/or its affiliates.

```
                                propertyValue = "MyQueue") // Ext. JNDI Name
  }
)

@Resources ({
  @Resource(name="targetCFRef",
            mappedName="MyTargetCF",   // External JNDI name
            type=jakarta.jms.ConnectionFactory.class),

  @Resource(name="targetDestRef",
            mappedName="MyTargetDest", // External JNDI name
            type=jakarta.jms.Destination.class)
})

public class MyMDB implements MessageListener {

  // inject a reference to the MDB context

  @Resource
  private MessageDrivenContext mdctx;

  // cache targetCF and targetDest for re-use (performance)

  private ConnectionFactory targetCF;
  private Destination targetDest;

  @TransactionAttribute(value = TransactionAttributeType.REQUIRED)
  public void onMessage(Message message) {

    System.out.println("My MDB got message: " + message);

    // Forward the message to "MyTargetDest" using "MyTargetCF"

    Connection jmsConnection = null;

    try {
      if (targetCF == null)
        targetCF = (jakarta.jms.ConnectionFactory)mdctx.lookup("targetCFRef");

      if (targetDest == null)
        targetDest = (jakarta.jms.Destination)mdctx.lookup("targetDestRef");

      jmsConnection = targetCF.createConnection();
      Session s = jmsConnection.createSession(false, Session.AUTO_ACKNOWLEDGE);
      MessageProducer mp = s.createProducer(null);

      mp.send(targetDest, message);

    } catch (JMSException e) {

      System.out.println("Forcing rollback due to exception " + e);
      e.printStackTrace();
      mdctx.setRollbackOnly();

    } finally {

      // Closing a connection automatically returns the connection and
      // its session plus producer to the resource reference pool.

      try { if (jmsConnection != null) jmsConnection.close(); }
      catch (JMSException ignored) {};
    }
```


Copyright © 2007, 2026, Oracle and/or its affiliates.

```
    }

}
```


Copyright © 2007, 2026, Oracle and/or its affiliates.

8

Migration and Recovery for Clustered MDBs

Learn about the migration and recovery for clustered MDBs. Oracle WebLogic Server supports migration and recovery for clustered JMS destinations. In the event of failure, you can bring a JMS destination back online on a different JVM. You can design your cluster so that when a server instance fails, it automatically migrates the JMS destination from the failed server in the cluster to an available server instance.
After a WebLogic JMS destination migrates, the EJB container attempts to reestablish the affected MDB connections so that message processing can resume when the destination becomes available.

> ⚠ **Caution**
>
> Service migration is not recommended for the following cases. In these cases, migration can result in either missing messages or duplicate message processing.
>
> Case 1, when all of the following are true:
>
> - The MDB `topicMessagesDistributionMode` is `One-Copy-Per-Server`
> - The MDB `distributedDestinationConnection` is `LocalOnly`
> - The MDB is `Durable`
> - The destination is configured as the logical name of a replicated distributed topic
>
> Case 2, when all of the following are true:
>
> - The MDB `topicMessagesDistributionMode` is `Compatibility`
> - The MDB is `Durable`
> - The destination is configured as the logical name of a distributed topic
>
> Case 3, when all of the following are true:
>
> - The MDB `topicMessagesDistributionMode` is `One-Copy-Per-Application`
> - The MDB `distributedDestinationConnection` is `LocalOnly`
> - The migration target server has no MDB instance. Best practice is to target MDB deployments to the entire cluster, to avoid this problem.
>
> For more information on topic message processing, see Configuring and Deploying MDBs Using JMS Topics .

> ⓘ **Note**
>
> A migratable service works with clustered servers only. A WebLogic JMS destination can migrate to another server within a cluster, but cannot migrate to a different cluster.

Copyright © 2007, 2026, Oracle and/or its affiliates.

MDBs can be targeted to clusters or individual Oracle WebLogic Server instances, but not to migratable targets. If an MDB is running in the same cluster as a migratable destination, you must ensure that MDB is deployed everywhere that its source destination may be hosted. You can do this in two ways:

- Deploy MDBs homogeneously to the cluster. (Recommended)
- Ensure that the MDB is deployed to every candidate server for each migratable target used by the JMS servers that host its source destination. If a migratable target has no constrained candidate-server list, all servers in its cluster are candidates. For more information on configuring migratable targets, see Understanding Migratable Target Servers in a Cluster in *Administering Clusters for Oracle WebLogic Server*.

For instructions on implementing the migratable service and for background information on WebLogic JMS migration and recovery services for clustered architectures, see JMS as a Migratable Service within a Cluster in *Administering JMS Resources for Oracle WebLogic Server*.

Copyright © 2007, 2026, Oracle and/or its affiliates.

9

Using Batching with Message-Driven Beans

Transaction batching allows a JMS MDB to process multiple messages in one container-managed transaction. Each message is processed by a separate `onMessage` invocation. Message receipts and participating transactional work from those invocations share the same transaction. This can reduce transaction-completion overhead. Different batches can be processed concurrently.
For information on transaction management within MDBs, see Configuring Transaction Management Strategy for an MDB.

> **Note**
>
> Transaction batching is not effective for all MDB applications. For example, database deadlocks can occur in an application where an MDB makes multiple calls to a database. Using the transaction batching feature can cause the MDB to lock more rows per transaction which can lead to database deadlocks.

This chapter includes the following sections:

- Configuring MDB Transaction Batching
- How MDB Transaction Batching Works

Configuring MDB Transaction Batching

Transaction batching applies to JMS MDBs that use container-managed transactions with the `Required` transaction attribute for `onMessage`. To enable batching, set `max-messages-in-transaction` within the `message-driven-descriptor` element of `weblogic-ejb-jar.xml`, or set the WebLogic-specific `maxMessagesInTransaction` activation configuration property, to a value greater than 1. The default value, 1, processes one message per transaction. `max-messages-in-transaction` specifies the maximum number of messages processed in one transaction. However, increasing the batch size can increase latency. You should start with a small value, 5 for example. You can increase this value as your application performance allows.

When using MDB batching, more messages are processed per transaction. This may cause more transactions to time out since more work is being performed in each transaction. To increase the transaction timeout, set the `trans-timeout-seconds` element within the MDB's `transaction-descriptor` element under `weblogic-enterprise-bean` in `weblogic-ejb-jar.xml`. Alternatively, apply WebLogic's `@TransactionTimeoutSeconds` annotation to the MDB class. For example, place the following annotation directly before the existing MDB class declaration, alongside its other class annotations, to configure a 60-second timeout:

```
@weblogic.javaee.TransactionTimeoutSeconds(60)
```

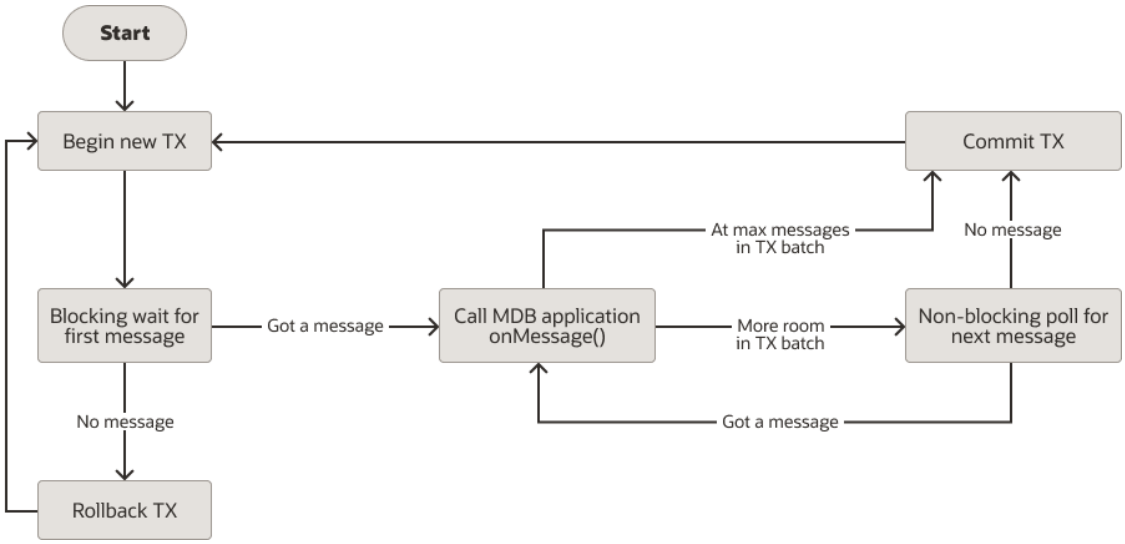
An explicitly configured `trans-timeout-seconds` value in `weblogic-ejb-jar.xml` takes precedence over the annotation.

Copyright © 2007, 2026, Oracle and/or its affiliates.

How MDB Transaction Batching Works

MDB transaction batching does not require any changes to application code. As far as the application is concerned, individual messages are still processed one by one. There is no application level message list.
For each batch, the container begins a transaction and invokes `onMessage` separately for each message as it is received. During normal processing, it attempts to commit the transaction when the batch limit is reached or a subsequent receive returns no message. A batch can therefore contain fewer messages than the maximum specified by `max-messages-in-transaction` or the `maxMessagesInTransaction` activation configuration property. Figure 9-1 illustrates the normal receive, process and commit sequence.

Figure 9-1 MDB Transaction Batching Transaction Processing Flow



If processing a message causes the batch transaction to roll back, all message receipts and participating transactional work in that transaction are rolled back together. During error recovery, the container can temporarily use one message per transaction before resuming batching. This can also occur after a transaction timeout. Messages rolled back with the transaction become eligible for redelivery according to the JMS provider's redelivery configuration and policies.

Copyright © 2007, 2026, Oracle and/or its affiliates.

10

Configuring and Deploying MDBs Using JMS Topics

Examine how to develop an MDB that automatically sets up JMS topic subscriptions and then processes subscription messages. A message that is published to a JMS topic is replicated to all subscriptions that have a matching selector filter.
A single deployed MDB may create multiple topic subscriptions and may have one or more free pools per host Oracle WebLogic Server instance. This behavior is controlled by MDB attribute settings, topic type, and whether the MDB is running on the same cluster or JVM as its topic. (For information about MDB free pools, see MDBs and Concurrent Processing.)

This chapter also describes how to use topic MDBs together with WebLogic JMS distributed topics. WebLogic JMS distributed topics are logical topics that are composed of multiple physical topics, where each physical topic is hosted on a different JMS Server instance. This distributed topic capability was significantly enhanced in Oracle WebLogic Server 10.3.4 to provide increased scalability and high availability. The enhancements include direct support for remotely hosted distributed topics, for fully distributing the processing of a single logical subscription across multiple physical subscriptions, and for multiple JVMs to process messages from the same physical subscription.

This chapter includes the following sections:

- Supported Topic Types
- The Most Commonly Used MDB Attributes
- Best Practices
- Configuring for Service Migration
- Upgrading Applications from Previous Releases
- Topic MDB Sample

For additional information about JMS topics see:

- Topic Deployment Scenarios
- Topic Subscription Identifiers
- How WebLogic MDBs Leverage WebLogic JMS Extensions
- Developing Advanced Pub/Sub Applications in *Developing JMS Applications for Oracle WebLogic Server*.
- Tuning WebLogic JMS in *Tuning Performance of Oracle WebLogic Server*
- Tuning Message-Driven Beans in *Tuning Performance of Oracle WebLogic Server*.

Oracle recommends reviewing the previous chapters of this book before reading this chapter.

Supported Topic Types

WebLogic MDBs support singleton, foreign provider, replicated distributed, and partitioned distributed topic types.
WebLogic MDBs support the following types of topics:

Copyright © 2007, 2026, Oracle and/or its affiliates.

- **Singleton topics** -- A singleton topic is either a non-distributed WebLogic JMS topic or a reference to a particular member topic of a WebLogic JMS distributed topic. The JNDI name syntax for a WebLogic JMS uniform distributed topic member is based on the name of the JMS server that hosts the member: *`jms-server-name@jndi-name-of-distributed-topic`*.
- **Foreign provider topics** -- Non-WebLogic JMS topics are called *foreign provider topics*. MDBs treat foreign provider topics similarly to singleton topics. Such topics are typically considered to be remote.
- **Replicated distributed topics** -- WebLogic JMS distributed topics are logical topics composed of multiple physical topics, where each physical topic is hosted on a different JMS server instance in the same cluster. In releases of Oracle WebLogic Server prior to 10.3.4, each message sent to any member of a distributed topic is always automatically replicated (forwarded) to all subscriptions on all of the other members of the distributed topic. This kind of distributed topic is still supported and is now called a *replicated distributed topic* (abbreviated *RDT*).
- **Partitioned distributed topics** - WebLogic JMS distributed topics are logical topics composed of multiple physical topics, where each physical topic is hosted on a different JMS server instance in the same cluster. A *partitioned distributed topic* (abbreviated *PDT*) does not forward messages between members. Messages published to a member of a PDT are only copied to subscriptions on that member. Partitioned distributed topics are supported starting with Oracle WebLogic Server 10.3.4.

To configure a distributed topic type, you set `Partitioned` or `Replicated` as the value for the JMS distributed topic configuration attribute `JMS Forwarding Policy`. See Configuring Partitioned Distributed Topics in *Administering JMS Resources for Oracle WebLogic Server*.

The Most Commonly Used MDB Attributes

Examine some of the topic MDB attributes.
The most commonly used topic MDB attributes are:

- JMS destination and connection factory
- Subscription durability
- Container managed transactions
- Message distribution tuning

Some other useful topic MDB attributes are:

- Free pool size
- Auto-delete on undeploy
- Message filtering (JMS selectors)
- Subscription identifier

The message distribution tuning settings include the `topicMessagesDistributionMode`, `distributedDestinationConnection`, and `generate-unique-jms-client-id` attributes.

Configure `generate-unique-jms-client-id` in the `message-driven-descriptor` element of `weblogic-ejb-jar.xml`. This setting controls whether WebLogic Server generates unique JMS client IDs in `Compatibility` mode. In the One-Copy modes, JMS client IDs are determined by the selected distribution mode's behavior.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Most attributes can be configured either by using an annotation or via descriptor XML stanzas. In addition, specific attribute names for descriptor XML stanzas and annotations are summarized in the tables in Deployment Elements and Annotations for MDBs.

Setting the JMS Destination, Destination Type, and Connection Factory

A topic MDB's configuration must properly specify the location of its JMS connection factory, its destination, and its destination type. Typically, this is accomplished by:

1. Specify the topic type. In an Enterprise Beans 4.0 `ejb-jar.xml` descriptor, set `message-destination-type` to `jakarta.jms.Topic` within the MDB's `message-driven` element. Alternatively, set the `destinationType` activation configuration property to `jakarta.jms.Topic` in `ejb-jar.xml` or in `@MessageDriven.activationConfig`.
2. Specifying a connection factory JNDI name and a destination JNDI name. Specifying a connection factory JNDI name is usually not necessary if the connection factory is hosted on the same cluster or server as the MDB. The default usually suffices.
3. If the destination is not located in the same cluster or server as the MDB pool, administratively configure a mapping from the remote destination and connection factory JNDI entries to local JNDI entries that match those specified in #2. There are alternative approaches to referencing remote resources, but the mapping approach is the Oracle-recommended best practice.

For each free pool, the MDB container creates a connection using the specified connection factory, then uses the connection to find or create one or more subscriptions on its destination, and finally uses the connection to create JMS consumers that receive the messages from the subscription(s).

For the specific names of connection factory and destination MDB attributes, as well as recommended JNDI mapping configuration, see Configuring MDBs for Destinations.

Setting Subscription Durability

MDBs automatically create subscriptions on JMS topics. JMS topics support two types of subscriptions: *durable* and *non-durable*.

- Non-durable subscriptions exist only for the length of time their subscribers exist. When a subscription has no more subscribers, the subscription is automatically deleted. Messages stored in a non-durable subscription are never recovered after a JMS server shut down or crash.
- Durable subscriptions make it possible for a subscriber to receive messages that are published while the subscriber application is unavailable. For each durable subscription on a topic, JMS retains matching persistent messages until they are acknowledged, unless they expire or are otherwise removed under the provider's message-management policies. This includes messages published while the durable subscription has no active consumer. For messages consumed in a transaction, acknowledgement takes effect when the transaction commits. JMS also stores a copy of each non-persistent message in each durable subscription, but such messages are not recovered if the JMS server shuts down or crashes.

  Non-durable subscriptions are the default. To configure a durable subscription, set the `subscriptionDurability` activation configuration property to `Durable`. In `ejb-jar.xml`,

Copyright © 2007, 2026, Oracle and/or its affiliates.

add the following property to the MDB's `activation-config` element within `message-driven`:

```
<activation-config-property>
  <activation-config-property-name>subscriptionDurability</activation-config-property-name>
  <activation-config-property-value>Durable</activation-config-property-value>
</activation-config-property>
```

Alternatively, add `@ActivationConfigProperty(propertyName = "subscriptionDurability", propertyValue = "Durable")` to `@MessageDriven.activationConfig`.

Setting Automatic Deletion of Durable Subscriptions

You can configure an MDB to automatically delete a durable topic subscription when the MDB is undeployed or deleted from a server. To configure an MDB to automatically delete durable topic subscriptions, set `durable-subscription-deletion` to `True`. By default, `durable-subscription-deletion` is set to `False`

Setting Container Managed Transactions

See Configuring Transaction Management Strategy for an MDB.

Setting Message Filtering (JMS Selectors)

JMS provides an SQL-like syntax for filtering messages based on standard JMS message header fields and message properties. In addition, WebLogic JMS supports an extension to the selector syntax that allows the specification of selectors that include XML "xpath" expressions for filtering XML messages based on their XML contents.

One way to specify a message selector is to specify it as the `propertyValue` for an `ActivationConfigProperty` with `propertyName = "messageSelector"`.

The syntax of JMS selectors is fully described in the Javadoc for the `jakarta.jms.Message` class. The WebLogic xpath selector extension syntax is described in Filtering Messages in *Developing JMS Applications for Oracle WebLogic Server*.

Controlling MDB Concurrency

As discussed in Topic Deployment Scenarios, an MDB deployment may create one or more MDB free pools. The `max-beans-in-free-pool` and `dispatch-policy descriptor` attributes work together to control MDB thread concurrency in an MDB free pool as follows:

- For a discussion of how to determine the number of concurrent MDBs, see Determining the Number of Concurrent MDBs in *Tuning Performance of Oracle WebLogic Server*.
- In `Compatibility` mode, concurrency also depends on the JMS provider and transaction-batching configuration. With WebLogic JMS and one message per transaction, transactional MDB invocations can run concurrently within the configured concurrency limits. With container-managed transaction batching in `Compatibility` mode, only one MDB `onMessage` invocation runs at a time per free pool. In addition, `max-beans-in-free-pool` should be explicitly set to `1` for bean-managed transaction MDBs that are driven by a foreign (non-WebLogic) topic.

Copyright © 2007, 2026, Oracle and/or its affiliates.

> ⚠ **Caution**
>
> **Non-transactional Foreign Topics:** Oracle recommends explicitly setting `max-beans-in-free-pool` to `1` for non-transactional MDBs that consume from foreign (non-WebLogic) topics through WebLogic Server's JMS integration. This integration supports foreign topics in `Compatibility` mode and limits message processing to one concurrent MDB invocation per free pool. This concurrency limit does not make non-transactional processing atomic or guarantee recovery from application failures.
>
> *Unit-of-Order:* Oracle recommends explicitly setting `max-beans-in-free-pool` to `1` for non-transactional `Compatibility` mode MDBs that consume from a WebLogic JMS topic and process messages that have a WebLogic JMS Unit-of-Order value. Unit-of-Order messages in this use case may not be processed in order unless `max-beans-in-free-pool` is set to `1`.

See Tuning Message-Driven Beans in *Tuning Performance of Oracle WebLogic Server* for more information.

Setting Subscription Identifiers

Individual JMS topic subscriptions are created and referenced based on their "subscription identifier," which an MDB generates based on a number of MDB configuration settings. For a discussion of the syntax of generated subscription identifiers, see Topic Subscription Identifiers.

Setting Message Distribution Tuning

This section describes how and when to use message distribution tuning settings. It contains information that applies to all topic types (singleton, foreign, and distributed). The settings include the `topicMessagesDistributionMode`, `distributedDestinationConnection`, and `generate-unique-jms-client-id` attributes. They control where topic subscriptions are created, what the subscription identifiers are, and whether an MDB processes each published topic message only once or once per server.

For detailed descriptions and diagrams of the resulting automatically generated subscription IDs, subscription locations, and deployed MDB free pool locations, see Topic Deployment Scenarios, and Topic Subscription Identifiers.

Setting topicMessagesDistributionMode

Use the `topicMessagesDistributionMode` setting in combination with the `distributedDestinationConnection` setting or the `generate-unique-jms-client-id` setting to control topic message processing behavior. To configure `topicMessagesDistributionMode`, set that activation configuration property's value in the MDB's `activation-config` element in `ejb-jar.xml`, or use `@ActivationConfigProperty(propertyName = "topicMessagesDistributionMode", propertyValue = "One-Copy-Per-Application")` within `@MessageDriven.activationConfig`, substituting the desired mode.

The valid values for `topicMessagesDistributionMode` are:

- **`One-Copy-Per-Application`** -- Specifies that the MDB application as a whole receives each message published to a distributed topic once, no matter how many servers host the

Copyright © 2007, 2026, Oracle and/or its affiliates.

application. This mode works with WebLogic JMS singleton and distributed topics in Oracle WebLogic Server 10.3.4 and later.

- **`One-Copy-Per-Server`** -- Specifies that each deployment instance of the MDB application receives every message published to a distributed topic. This mode works with WebLogic JMS singleton and distributed topics in Oracle WebLogic Server 10.3.4 and later.
- **`Compatibility`** - (Default) Specifies that the MDB application handles messages from distributed topics in the same way they were handled in Oracle WebLogic Server releases prior to 10.3.4. The mode supports durable and non-durable subscriptions with foreign (non-WebLogic) topics, local replicated distributed topics (RDTs), and singleton WebLogic topics; it also supports non-durable subscriptions with remote replicated distributed topics. See the Compatibility notes section for more detail.

> **Note**
>
> Oracle recommends using the `One-Copy-Per-Application` and `One-Copy-Per-Server` modes for most new applications, except those that must consume from WebLogic JMS topics in versions of Oracle WebLogic Server prior to 10.3.4 or from foreign (non-WebLogic) topics.

The topic distribution modes support different topic types and versions with the following restrictions:

- The `One-Copy-Per-Application` and `One-Copy-Per-Server` modes work only with WebLogic singleton and distributed topics in Oracle WebLogic Server 10.3.4 and later. WebLogic MDBs log a warning and do not process messages with these modes when using a foreign (non-WebLogic) topic or when using a WebLogic topic from Oracle WebLogic Server releases prior to 10.3.4.
- `One-Copy-Per-Server` topic MDBs that are durable, that subscribe to a *local* RDT, and that use the default `LocalOnly` value for the `distributedDestinationConnection` attribute, do not support Service Migration and require that exactly one topic member be configured per Oracle WebLogic Server instance. If a service migration occurs, if there is no local topic member configured, or if more than one topic member is deployed per server, then the application may experience duplicate or lost messages and may also create abandoned subscriptions that accumulate unprocessed messages. If service migration is required, then use the `EveryMember` option for the `distributedDestinationConnection` attribute instead of the default `LocalOnly`. All other JMS service-migration and MDB deployment requirements still apply. See Migration and Recovery for Clustered MDBs.
- The `Compatibility` mode supports durable and non-durable subscriptions with foreign (non-WebLogic) topics, with local replicated distributed topics (RDTs) (with limitations described later), and with singleton WebLogic topics. Compatibility mode also supports non-durable subscriptions with a remote RDT. A deployment of a durable MDB that subscribes to the logical JNDI name of a remote RDT may succeed, but the MDB deployment will fail to connect, with `Warning` log messages. Similarly, a deployment of an MDB that subscribes to a WebLogic PDT may succeed, but the MDB deployment will fail to connect, with `Warning` log messages. See Notes on the Compatibility mode of topicMessagesDistributionMode, for more detail.
- For durable Compatibility-mode MDBs that subscribe to a local RDT, see Notes on the Compatibility mode of topicMessagesDistributionMode.

Copyright © 2007, 2026, Oracle and/or its affiliates.

For detailed descriptions and diagrams of MDB generated subscriptions, subscription IDs, and free pool locations, refer to Topic Deployment Scenarios, and Topic Subscription Identifiers.

Setting distributedDestinationConnection

To optionally fine tune the behavior of the `One-Copy-Per-Application` and `One-Copy-Per-Server` modes of `topicMessagesDistributionMode` for a local distributed topic, you can use the `distributedDestinationConnection` activation config property. Alternatively, you can use the `distributed-destination-connection` element in the `weblogic-ejb-jar.xml` deployment descriptor. The valid values are `LocalOnly` and `EveryMember`.

For local distributed topics, `EveryMember` creates an MDB free pool for each topic member, while `LocalOnly` restricts connections to members on the MDB's current WebLogic Server. Although the descriptor default is `LocalOnly`, a local partitioned distributed topic in `One-Copy-Per-Server` mode uses `EveryMember`. When `LocalOnly` is explicitly configured or supplied by default for that combination, WebLogic overrides it and logs warning `015035`.

Use this setting to tune local distributed-topic connections in the One-Copy modes. For remote distributed topics, the container determines member connections from the topic type, distribution mode and subscription durability. In `Compatibility` mode, this setting does not control which destination members the MDB connects to, but it can affect generated JMS client IDs when `generate-unique-jms-client-id` is enabled.

One reason to use `EveryMember` is that the `LocalOnly` option for durable MDBs has restrictions for local RDTs in the `One-Copy-Per-Server` mode. See Warning About Using Local RDTs with Durable MDBs.

Another reason to use `EveryMember` is to better handle uneven message loads or message processing delays. See Handling Uneven Message Loads and/or Message Processing Delays, for advice.

Notes on the Compatibility mode of topicMessagesDistributionMode

- See Setting topicMessagesDistributionMode, for a statement about supported topic types and versions.
- If you're using the `Compatibility` `topicMessagesDistributionMode` in combination with non-transactional MDBs, and the topic is a foreign (non-WebLogic) destination, or the topic is a WebLogic destination with Unit-of-Order (UOO) messages, then see Controlling MDB Concurrency, for warnings.
- Set the `generate-unique-jms-client-id` attribute to change behavior:
  - If `generate-unique-jms-client-id` is set to true, each durable MDB free pool generates a unique subscriber ID. Each MDB free pool will then receive a copy of each published message. For more information see Topic Subscription Identifiers. For more information about free pools, see MDBs and Concurrent Processing, and Topic Deployment Scenarios.
  - If `generate-unique-jms-client-id` is false (the default), only one subscription will be created by a durable MDB, and only one MDB free pool will successfully connect to the durable subscription (the remaining MDB pools will fail to connect, log warnings, and will keep retrying).
- For durable subscription MDBs that subscribe to the logical name of a local replicated distributed topic (a local RDT), only the configuration described in Warning About Using Local RDTs with Durable MDBs, is supported.

Copyright © 2007, 2026, Oracle and/or its affiliates.

- For durable subscription MDBs that subscribe to the logical name of a remote replicated topic (a remote RDT):
  - A deployment of a durable MDB that subscribes to the logical JNDI name of the RDT may succeed, but the MDB will fail to connect and will log warning messages.
- For durable subscription MDBs that subscribe to a particular member destination of a remote replicated topic:
  - A deployment of a durable MDB that subscribes directly to a member of the RDT will succeed, and the subsequent behavior will be determined by the `generate-unique-jms-client-id` setting. For a uniform distributed destination, the JNDI name of a particular member is `"jms-server-name@udd-jndi-name"`.
- For a non-durable subscription MDB that subscribes to the logical name of a local replicated distributed topic (a local RDT), only the configuration described in Warning About Using Local RDTs with Non-Durable MDBs, is fully supported.
- In `Compatibility` mode, `distributedDestinationConnection` does not control which destination members the MDB connects to, but it can affect generated JMS client IDs when `generate-unique-jms-client-id` is enabled.

Best Practices

Examine some of the best practices when you configure MDBs.
Consider the information in the following sections to help you configure MDBs.

Warning about Non-Transactional MDBs in Compatibility Mode

If you're using the `Compatibility` mode of `topicMessagesDistributionMode` in combination with non-transactional MDBs and the topic is a foreign (non-WebLogic) destination or the topic is a WebLogic destination with Unit-of-Order (UOO) messages, see Controlling MDB Concurrency, for warnings.

Warning About Using Local RDTs with Durable MDBs

In `Compatibility` mode, for durable subscription MDBs that subscribe to the logical name of a local replicated distributed topic (a local RDT), only the following configuration is supported:

- Always set `generate-unique-jms-client-id` to true.
- Ensure each Oracle WebLogic Server in the cluster hosts exactly one member of the RDT.
- Do not use WebLogic JMS `service-migration`. It is unsupported for this use case; but you can use "whole server migration."
- Note that each server receives a copy of each message sent to the topic. When a message arrives at one of the RDT physical topic members, the RDT automatically ensures that a copy of the message is forwarded to each of the other members of the topic.

For durable MDBs in `One-Copy-Per-Server` mode that subscribe to the logical name of a local RDT with `distributedDestinationConnection=LocalOnly`, the same one-member-per-server and service-migration restrictions apply. In this mode, each server hosting the MDB uses its own subscription identity. The `generate-unique-jms-client-id` setting applies to `Compatibility` mode and is not required for this One-Copy mode.

Copyright © 2007, 2026, Oracle and/or its affiliates.

If your configuration does not match the recommendations, you may get nondeterministic behavior, including lost messages, duplicate messages, and stuck messages. For more information, including alternatives, see Setting topicMessagesDistributionMode.

Warning About Using Local RDTs with Non-Durable MDBs

In `Compatibility` mode, for non-durable subscription MDBs that subscribe to the logical name of a local replicated distributed topic (a local RDT), only the following configuration is fully supported:

- Ensure each Oracle WebLogic Server in the cluster hosts exactly one member of the RDT.
- Do not use WebLogic JMS `service-migration`. It is unsupported for this use case; but you can use "whole server migration."
- Note that each server receives a copy of each message sent to the topic. When a message arrives at one of the RDT physical topic members, the RDT automatically ensures that a copy of the message is forwarded to each of the other members of the topic.

The same one-member-per-server and service-migration restrictions apply to non-durable MDBs in `One-Copy-Per-Server` mode that subscribe to the logical name of a local RDT with `distributedDestinationConnection=LocalOnly`. In this mode, each server hosting the MDB uses its own subscription identity.

If your configuration does not match the recommendations, you may get nondeterministic behavior, including lost messages, duplicate messages, and stuck messages. For more information, including alternatives, see Setting topicMessagesDistributionMode.

Warning about Changing Durable MDB Attributes, Topic Type, EJB Name

Changing MDB or JMS settings can cause the current messages on durable subscriptions to be deleted, or can cause existing durable subscriptions to be abandoned, deleted, or replaced in favor of new durable subscriptions. These settings include topic type, JMS selector, distribution tuning, subscription durability, `ejb-name`, and `client-id`.

Abandoned durable subscriptions continue to accumulate messages even though no MDB is processing the messages. This can eventually lead to quota exceptions or even JVM out-of-memory errors that prevent additional messages from being published to the topic.

For a discussion about locating and removing abandoned subscriptions see Managing Durable Subscriptions in *Developing JMS Applications for Oracle WebLogic Server*. For a discussion about subscription IDs and locations, see Topic Subscription Identifiers.

Choosing Between Partitioned and Replicated Topics

Supported Topic Types, describes the two types of WebLogic distributed topics (partitioned and replicated). In general, Oracle recommends using partitioned topics (PDTs), when available, except for these two cases:

- When replicated topic (RDT) behavior is required to interoperate with legacy applications or non-MDB applications.
- In the local RDT case in the `One-Copy-Per-Server LocalOnly` case under certain message loads. The message load determines whether the heavy forwarding overhead built into an RDT is less expensive in comparison to the increased network traffic required for the fully connected topology in the PDT `One-Copy-Per-Server` mode. In general, it is better to use a PDT for non-persistent or "lighter" persistent message loads.

Copyright © 2007, 2026, Oracle and/or its affiliates.

To configure a distributed topic type, you set `Partitioned` or `Replicated` as the value for the WebLogic JMS Distributed Topic configuration attribute JMS Forwarding Policy. See Configuring Partitioned Distributed Topics in *Administering JMS Resources for Oracle WebLogic Server*.

Choosing an MDB Topic Messages Distribution Mode

Oracle recommends using the `One-Copy-Per-Application` and `One-Copy-Per-Server` modes for most new applications, except for those that must consume from WebLogic JMS topics in Oracle WebLogic Server releases prior to 10.3.4 or from foreign (non-WebLogic) topics. These two modes only work with WebLogic JMS topics in Oracle WebLogic Server 10.3.4 or later.

Managing and Viewing Subscriptions:

See Topic Deployment Scenarios, and Topic Subscription Identifiers, for detailed discussions of the names and location of subscriptions.

See also Managing Durable Subscriptions in *Developing JMS Applications for Oracle WebLogic Server*.

Handling Uneven Message Loads and/or Message Processing Delays

For applications with uneven message loads or unanticipated message processing delays, you may want to consider the following:

- In `One-Copy-Per-Application` mode with a local distributed topic, consider `distributedDestinationConnection=EveryMember` when message backlogs or processing costs are uneven across members. MDB instances on different servers can then participate in processing messages from shared subscriptions, although this can increase network traffic. In `One-Copy-Per-Server` mode, each MDB server uses its own subscription identity; `EveryMember` does not turn those separate subscriptions into one shared application subscription. For a PDT in that mode, `EveryMember` is already the effective connection mode.
- To distribute incoming messages among PDT members, configure producer load balancing on the producer's connection factory. Enabling `Load Balancing Enabled` and disabling `Server Affinity Enabled` can spread sends beyond local members, but the result also depends on the distributed topic's load-balancing policy and routing constraints such as transaction affinity and Unit-of-Order. These settings do not guarantee equal MDB processing load. Disabling `Server Affinity Enabled` does not disable the RDT producer load balancer's local-member preference. See Load Balancing Messages Across a Distributed Destination in *Administering JMS Resources for Oracle WebLogic Server*.
- Decrease the WebLogic JMS asynchronous message pipeline size to `1` to prevent additional messages from being pushed to an MDB thread that is already blocked processing a previous message. The default for this setting is `10`, and it is configured by (a) configuring a custom WebLogic connection factory with the Messages Maximum attribute tuned to `1` and XA Enabled set to true, (b) targeting the connection factory to the same cluster that hosts the distributed topic, and (c) modifying the MDB so that it references the custom connection factory.

Configuring for Service Migration

Examine how to configure service migration manually or automatically in an MDB deployment.

Copyright © 2007, 2026, Oracle and/or its affiliates.

For durable MDBs in `One-Copy-Per-Server` mode that subscribe to a local RDT, configure `distributedDestinationConnection=EveryMember` if JMS service migration is required. All other JMS service-migration and MDB deployment requirements still apply. The `LocalOnly` configuration is subject to the restrictions in Warning About Using Local RDTs with Durable MDBs. `Compatibility` mode has separate restrictions described in that section; setting `EveryMember` does not remove them.

With `LocalOnly`, member connections depend on which topic members are hosted on the MDB's current server. For `One-Copy-Per-Application`, ensure that the MDB is deployed on every candidate server that can host a migrated topic member. See Migration and Recovery for Clustered MDBs for the migration cautions and deployment guidance. Whole server migration remains an alternative for the configurations described in the local-RDT warnings.

Upgrading Applications from Previous Releases

Examine the steps to upgrade MDB applications to take advantage of features such as scalability and high availability in the latest releases.
As described throughout this chapter, new JMS features in Oracle WebLogic Server 10.3.4, such as relaxed client ID, sharable subscriptions, and partitioned distributed topics, make it possible to implement and deploy MDBs that provide enhanced scalability and high availability. To take advantage of these features, you must upgrade MDB applications written for releases of Oracle WebLogic Server prior to 10.3.4.

`Compatibility`, the default setting for `topicMessagesDistributionMode`, preserves legacy topic-message distribution behavior. Applications from earlier releases must also satisfy the application-migration requirements of the target WebLogic Server release, including any required migration from Java EE `javax.*` APIs to their Jakarta EE `jakarta.*` equivalents. See Setting topicMessagesDistributionMode for the behavior of this mode.

To upgrade applications from previous releases,

1. Consider changing to a partitioned distributed topic. See Choosing Between Partitioned and Replicated Topics.
2. Set the `topicMessagesDistributionMode` to `One-Copy-Per-Server` or `One-Copy-Per-Application` and tune the `distributedDestinationConnection` options. See Setting Message Distribution Tuning.

> ⚠ **Caution**
>
> Current messages are not preserved when changing out of `Compatibility` mode. See Warning about Changing Durable MDB Attributes, Topic Type, EJB Name.

Topic MDB Sample

Examine a sample MDB using distributed topics.
Example 10-1 shows an MDB that uses a durable subscription to a WebLogic JMS topic, processes messages in a container-managed transaction, and forwards them to a target destination.

The MDB connects using JMS connection factory `MyCF` to receive from topic `MyTopic`. It forwards the messages to `MyTargetDest` using a connection generated from connection factory `MyTargetCF`.

Copyright © 2007, 2026, Oracle and/or its affiliates.

This example assumes that `MyTopic`, `MyCF`, `MyTargetCF` and `MyTargetDest` are configured and available in the MDB's local JNDI tree. For WebLogic JMS connection factories defined in a JMS module, enable `XAConnectionFactoryEnabled` on both `MyCF` and `MyTargetCF` to support transactional receipt and forwarding. For more about MDB connection-factory configuration, see How to Set connection-factory-jndi-name.

The `Required` transaction attribute causes `onMessage` to execute in a container-managed transaction. Message receipt and JMS operations that participate in that transaction are committed or rolled back together. The call to `createSession(false, Session.AUTO_ACKNOWLEDGE)` does not, by itself, exclude a WebLogic-managed JMS session from participating in the container transaction.

Resource reference pooling note: The MDB uses a resource reference to access `MyTargetCF`. The resource reference automatically enables JMS producer pooling, as described in Enhanced Support for Using WebLogic JMS with EJBs and Servlets in *Developing JMS Applications for Oracle WebLogic Server*.

For a similar sample using queues instead of topics, see Example 7-3.

Example 10-1 Sample MDB Using Distributed Topics

```
package test;
import jakarta.annotation.Resources;
import jakarta.annotation.Resource;
import jakarta.ejb.ActivationConfigProperty;
import jakarta.ejb.MessageDriven;
import jakarta.ejb.MessageDrivenContext;
import jakarta.ejb.TransactionAttribute;
import jakarta.ejb.TransactionAttributeType;
import jakarta.jms.*;

@MessageDriven(
  name = "MyMDB",
  activationConfig = {
    @ActivationConfigProperty(propertyName  = "destinationType",
                              propertyValue = "jakarta.jms.Topic"),

    @ActivationConfigProperty(propertyName  = "subscriptionDurability",
                              propertyValue = "Durable"),

    @ActivationConfigProperty(propertyName  = "connectionFactoryJndiName",
                              propertyValue = "MyCF"), // External JNDI Name

    @ActivationConfigProperty(propertyName  = "destinationJndiName",
                              propertyValue = "MyTopic"), // Ext. JNDI Name

    @ActivationConfigProperty(propertyName  = "topicMessagesDistributionMode",
                              propertyValue = "One-Copy-Per-Application")
  }
)

@Resources ({
  @Resource(name="targetCFRef",
            mappedName="MyTargetCF",   // External JNDI name
            type=jakarta.jms.ConnectionFactory.class),

  @Resource(name="targetDestRef",
            mappedName="MyTargetDest", // External JNDI name
            type=jakarta.jms.Destination.class)
})

public class MyMDB implements MessageListener {
```


Copyright © 2007, 2026, Oracle and/or its affiliates.

```
  // inject a reference to the MDB context

  @Resource
  private MessageDrivenContext mdctx;

  // cache targetCF and targetDest for re-use (performance)

  private ConnectionFactory targetCF;
  private Destination targetDest;

  @TransactionAttribute(value = TransactionAttributeType.REQUIRED)
  public void onMessage(Message message) {

    System.out.println("My MDB got message: " + message);

    // Forward the message to "MyTargetDest" using "MyTargetCF"

    Connection jmsConnection = null;

    try {
      if (targetCF == null)
        targetCF = (jakarta.jms.ConnectionFactory)mdctx.lookup("targetCFRef");

      if (targetDest == null)
        targetDest = (jakarta.jms.Destination)mdctx.lookup("targetDestRef");

      jmsConnection = targetCF.createConnection();
      Session s = jmsConnection.createSession(false, Session.AUTO_ACKNOWLEDGE);
      MessageProducer mp = s.createProducer(null);

      mp.send(targetDest, message);

    } catch (JMSException e) {

      System.out.println("Forcing rollback due to exception " + e);
      e.printStackTrace();
      mdctx.setRollbackOnly();

    } finally {

      // Closing a connection automatically returns the connection and
      // its session plus producer to the resource reference pool.

      try { if (jmsConnection != null) jmsConnection.close(); }
      catch (JMSException ignored) {};
    }
  }

}
```


Copyright © 2007, 2026, Oracle and/or its affiliates.

11

Deployment Elements and Annotations for MDBs

Examine the deployment elements and configuration properties that affect the behavior of MDBs.
Each row in Table 11-1 describes a deployment element (used in an EJB deployment descriptor) and its associated configuration property (specified in annotations). Not all elements have associated properties.

For information about using deployment descriptors versus using annotations in MDBs, see Programming EJB 4.0 Compliant MDBs.

> **Note**
>
> For those elements that have an associated configuration property, Oracle recommends that you use that property instead of the element.

Table 11-1 is organized as follows:

- Each element in the **Element** column is followed in parentheses by the name of the deployment descriptor in which the element is used. Elements in the `weblogic-ejb-jar.xml` descriptor link to a more complete explanation of the element in weblogic-ejb-jar.xml Deployment Descriptor Reference in *Developing Enterprise Beans, Version 4.0, for Oracle WebLogic Server*. For elements in `ejb-jar.xml`, see the schema at `https://jakarta.ee/xml/ns/jakartaee/ejb-jar_4_0.xsd`.
- Unless otherwise noted, all the properties listed in the **Configuration Property** column are activation configuration properties, that is, properties defined using `@ActivationConfigProperty` annotations or using an `activation-config-property` element in the `message-driven` stanza of an `ejb-jar.xml` descriptor. For more information about using `@ActivationConfigProperty`, see Programming EJB 4.0 Compliant MDBs.

  > **Note**
  >
  > Based on the Enterprise Bean 4.0 specification, the `jakarta.ejb.ActivationConfigProperty` annotation is used for MDBs only. This annotation is not used for session or entity beans.

- The **Configuration Property** column also lists annotations and properties that are not `@ActivationConfigProperty` properties. In those cases, the property is followed by the name of the annotation and by the `import` statement required for using that annotation.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Table 11-1 Deployment Elements and Annotations for MDBs

Element	Configuration Property	Description	Allowable Values	Default
`activation-config-property` (`ejb-jar.xml`)	`acknowledgeMode`	Specifies the JMS acknowledgement mode for messages received outside a transaction. For container-managed message delivery with the Required transaction attribute, acknowledgement is controlled by transaction completion.	`Auto-acknowledge` — Automatically acknowledges messages according to the provider's nontransactional delivery semantics, typically after the message has been successfully processed. `Dups-ok-acknowledge` — Allows lazy acknowledgement, which can result in duplicate message delivery.	`Auto-acknowledge`
`connection-factory-jndi-name` (`weblogic-ejb-jar.xml`)	`connectionFactoryLookup`	The lookup name of a JMS connection factory that will be used to connect to the JMS provider from which a JMS message-driven bean is to receive messages.	Valid lookup name	`weblogic.jms.MessageDrivenBeanConnectionFactory`
`connection-factory-jndi-name` (`weblogic-ejb-jar.xml`)	`connectionFactoryJndiName`	The JNDI name of the JMS connection factory that the MDB uses to create connections to the JMS provider. See How to Set connection-factory-jndi-name.	Valid JNDI name	`weblogic.jms.MessageDrivenBeanConnectionFactory`
`connection-factory-resource-link` (`weblogic-ejb-jar.xml`)	`connectionFactoryResourceLink`	Identifies an existing connection factory in a packaged JMS application module. For existing applications using these modules, see Using Application-Scoped JMS Resources.	`JMSModuleName#ResourceName`, where `JMSModuleName` matches the module's `name` in `weblogic-application.xml`, and `ResourceName` matches the resource's `name` in that JMS module.	n/a
`destination-jndi-name` (`weblogic-ejb-jar.xml`)	`destinationLookup`	The lookup name of a JMS queue or topic from which a JMS message-driven bean is to receive messages.	Valid lookup name	n/a
`destination-jndi-name` (`weblogic-ejb-jar.xml`)	`destinationJndiName`	The JNDI name used to associate an MDB with an actual JMS queue or topic deployed in the Oracle WebLogic Server JNDI tree. See How to Set destination-jndi-name.	Valid JNDI name	n/a

Copyright © 2007, 2026, Oracle and/or its affiliates.

Table 11-1 (Cont.) Deployment Elements and Annotations for MDBs

Element	Configuration Property	Description	Allowable Values	Default
`destination-resource-link` (`weblogic-ejb-jar.xml`)	`destinationResourceLink`	Identifies an existing queue or topic in a packaged JMS application module. For existing applications using these modules, see Using Application-Scoped JMS Resources.	`JMSModuleName#ResourceName`, where `JMSModuleName` matches the module's `name` in `weblogic-application.xml`, and `ResourceName` matches the resource's `name` in that JMS module.	n/a
`dispatch-policy` (`weblogic-ejb-jar.xml`)	n/a	This optional element allows you to specify a particular WorkManager for the bean. See Tuning Message-Driven Beans in *Tuning Performance of Oracle WebLogic Server*.	A configured Work Manager name or a legacy execute-queue name.	n/a
`distributed-destination-connection` (`weblogic-ejb-jar.xml`)	`distributedDestinationConnection`	Specifies whether an MDB that accesses a WebLogic JMS distributed destination (topic or queue) in the same cluster consumes from all distributed destination members or only those members local to the current Oracle WebLogic Server instance. May not apply to all use cases. See JMS Distributed Destinations, and Configuring and Deploying MDBs Using JMS Topics .	• `LocalOnly` • `EveryMember`	`LocalOnly`
`durable-subscription-deletion` (`weblogic-ejb-jar.xml`)	`durableSubscriptionDeletion`	Indicates whether you want durable topic subscriptions to be automatically deleted when an MDB is undeployed or removed.	• `True` • `False`	`False`
`generate-unique-jms-client-id` (`weblogic-ejb-jar.xml`)	`generateUniqueID` member of `@weblogic.javaee.JMSClientID`, for example, `@weblogic.javaee.JMSClientID(value = "MyClient", generateUniqueID = true)`.	Controls whether WebLogic generates unique JMS client IDs for MDB connections in Compatibility mode. It does not assign a separate client ID to each pooled MDB instance. In the One-Copy modes, JMS client IDs are determined by the selected distribution mode's behavior. See Topic Subscription Identifiers.	• `True` • `False`	`False`

Copyright © 2007, 2026, Oracle and/or its affiliates.

Table 11-1 (Cont.) Deployment Elements and Annotations for MDBs

Element	Configuration Property	Description	Allowable Values	Default
`initial-beans-in-free-pool` (`weblogic-ejb-jar.xml`)	n/a	Sets the initial size of the free pool. Oracle WebLogic Server populates the free pool with the specified number of bean instances for every bean class at startup. Populating the free pool in this way improves initial response time for the MDB, because initial requests for the bean can be satisfied without generating a new instance.	WebLogic descriptor: `0` through 2147483647	0
`initial-context-factory` (`weblogic-ejb-jar.xml`)	`initialContextFactory`	The initial context factory used to create the JNDI context in which the container looks up the JMS connection factory and destination. See How to Set initial-context-factory.	Valid name of an initial context factory	`weblogic.jndi.WLInitialContextFactory`
`init-suspend-seconds` (`weblogic-ejb-jar.xml`)	`initSuspendSeconds`	The initial number of seconds to suspend an MDB's JMS connection when the EJB container detects a JMS resource outage. See Configuring Suspension of Message Delivery During JMS Resource Outages.	WebLogic descriptor: non-negative integer, `0` through 2147483647	5
`jms-client-id` (`weblogic-ejb-jar.xml`)	`clientId`	The client identifier that will be used when connecting to the JMS provider from which a JMS message-driven bean is to receive messages.	n/a	Depends on the `topicMessagesDistributionMode` activation config property and possibly on `generate-unique-jms-client-id`. See Topic Subscription Identifiers.
`jms-client-id` (`weblogic-ejb-jar.xml`)	`jmsClientId`	WebLogic-specific configuration of the MDB's JMS client ID. Its use and the effective ID depend on the topic distribution mode. See Topic Subscription Identifiers.	n/a	Depends on the `topicMessagesDistributionMode` activation config property and possibly on `generate-unique-jms-client-id`. See Topic Subscription Identifiers.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Table 11-1 (Cont.) Deployment Elements and Annotations for MDBs

Element	Configuration Property	Description	Allowable Values	Default
`jms-polling-interval-seconds` (`weblogic-ejb-jar.xml`)	`jmsPollingIntervalSeconds`	The number of seconds between attempts by the EJB container to reconnect to a JMS destination that has become unavailable. See Migration and Recovery for Clustered MDBs.	WebLogic descriptor: non-negative integer, `0` through 2147483647	`10` seconds
`max-beans-in-free-pool` (`weblogic-ejb-jar.xml`)	n/a	The maximum number of bean instances in an MDB free pool. The actual number of instances is also limited by thread pool size as well as other factors. See Tuning Message-Driven Beans in *Tuning Performance of Oracle WebLogic Server*.	WebLogic descriptor: `0` through 2147483647	1000
`max-messages-in-transaction` (`weblogic-ejb-jar.xml`)	`maxMessagesInTransaction`	Specifies the maximum number of messages that can be in a transaction for this MDB.	WebLogic descriptor: positive integer, 1 through 2147483647	`1`
`max-suspend-seconds` (`weblogic-ejb-jar.xml`)	`maxSuspendSeconds`	The maximum number of seconds to suspend an MDB's JMS connection when the EJB container detects a JMS resource outage. See Configuring Suspension of Message Delivery During JMS Resource Outages.	WebLogic descriptor: non-negative integer, `0` through 2147483647	`60`
`message-destination-type` (`ejb-jar.xml`)	`destinationType`	Specifies the type of the JMS destination—the Java interface expected to be implemented by the destination.	• `jakarta.jms.Queue` • `jakarta.jms.Topic`	n/a

Copyright © 2007, 2026, Oracle and/or its affiliates.

Table 11-1 (Cont.) Deployment Elements and Annotations for MDBs

Element	Configuration Property	Description	Allowable Values	Default
`activation-config-property (ejb-jar.xml)`	`messagesMaximum`	Overrides the WebLogic JMS session's Messages Maximum setting, which limits the number of prefetched messages awaiting delivery to the MDB listener. The limit applies to asynchronous delivery and to synchronous prefetch when enabled. At the limit, further messages are retained on the server until the client can accept them. This property is ignored for foreign JMS providers.	`-1` for no configured message-count limit, or 1 through 2147483647. Do not specify `0`.	No MDB override. The session uses the JMS connection factory's Messages Maximum setting; the WebLogic connection-factory default is `10`.
`activation-config-property (ejb-jar.xml)`	`messageSelector`	A string used by a client to specify, by header field references and property references, the messages it is interested in. Only messages whose header and property values match the selector are delivered	Conditional expression using message properties, or message header	Null
`messaging-type (ejb-jar.xml)`	`messageListenerInterface` member of `@jakarta.ejb.MessageDriven`, for example, `@jakarta.ejb.MessageDriven(messageListenerInterface = jakarta.jms.MessageListener.class)`.	Identifies the message listener interface implemented by the MDB. For JMS delivery, this is `jakarta.jms.MessageListener`; resource-adapter delivery can use the listener interface supported by that adapter.	The fully qualified name of a supported message listener interface.	Inferred from the bean class when the message listener interface can be determined unambiguously. Otherwise, specify the interface explicitly.
`provider-url (weblogic-ejb-jar.xml)`	`providerURL`	The provider URL used to create the JNDI context for the JMS resource lookups, for example, `t3://host:port` for WebLogic Server. The URL syntax depends on the JNDI provider. See How to Set provider-url.	Valid URL	Null
`resource-adapter-jndi-name (weblogic-ejb-jar.xml)`	`resourceAdapterJndiName`	For JCA-driven MDBs, identifies the resource adapter from which this MDB receives messages.	n/a	n/a

Copyright © 2007, 2026, Oracle and/or its affiliates.

Table 11-1 (Cont.) Deployment Elements and Annotations for MDBs

Element	Configuration Property	Description	Allowable Values	Default
`security-role-assignment` (`weblogic-ejb-jar.xml`)	n/a	Maps application roles in the `ejb-jar.xml` file to the names of security principals available in Oracle WebLogic Server.	n/a	n/a
`start-mdbs-with-application` (`weblogic-application.xml`)	n/a	Controls when MDBs start processing messages. When set to `true`, an MDB starts processing messages as soon as it is deployed, even if Oracle WebLogic Server has not completed booting. This can cause an MDB application to access uninitialized services or applications during boot up and, therefore, to fail. Set to false to defer message processing until after Oracle WebLogic Server opens its listen port.	• `True` • `False`	`False`
`activation-config-property` (`ejb-jar.xml`)	`subscriptionDurability`	Specifies whether a JMS topic subscription is `Durable` or `NonDurable`. For more information, see Setting Subscription Durability.	• `Durable` • `NonDurable`	`NonDurable`
`activation-config-property` (`ejb-jar.xml`)	`subscriptionName`	Specifies the topic subscription name. WebLogic's direct JMS MDB integration uses this property for durable subscriptions. For resource-adapter delivery, supported use depends on the resource adapter.	A subscription-name string.	For durable subscriptions in WebLogic's direct JMS integration, the name is derived from the MDB-generated client-ID expression in Compatibility mode and from the EJB name in the One-Copy modes. See Topic Subscription Identifiers. For resource-adapter delivery, consult the adapter's configuration.
`activation-config-property` (`ejb-jar.xml`)	`topicMessagesDistributionMode`	Sets the distribution mode for topic messages. See Configuring and Deploying MDBs Using JMS Topics .	• `One-Copy-Per-Application` • `One-Copy-Per-Server` • `Compatibility`	`Compatibility`

Copyright © 2007, 2026, Oracle and/or its affiliates.

Table 11-1 (Cont.) Deployment Elements and Annotations for MDBs

Element	Configuration Property	Description	Allowable Values	Default
`transaction-type` (`ejb-jar.xml`)	`value` member of `@TransactionManagement` (`TransactionManagementType.BEAN` or `TransactionManagementType.CONTAINER`). Example on the MDB class: `@jakarta.ejb.TransactionManagement(jakarta.ejb.TransactionManagementType.CONTAINER)`	Specifies an enterprise bean's transaction management type. For more information, see Configuring Transaction Management Strategy for an MDB.	• `Bean` • `Container`	`Container`
`trans-attribute` (`ejb-jar.xml`)	`value` member of `@TransactionAttribute`. Example on the MDB's message listener method: `@jakarta.ejb.TransactionAttribute(jakarta.ejb.TransactionAttributeType.REQUIRED)`	Specifies how the container must manage the transaction boundaries when delegating a method invocation to an enterprise bean's business method. **Note:** If the bean is specified as using container-managed transaction demarcation, either the `REQUIRED` or the `NOT_SUPPORTED` transaction attribute must be used for the message listener methods, and either the `REQUIRED`, `REQUIRES_NEW`, or the `NOT_SUPPORTED` transaction attribute for time-out callback methods. For more information, see Configuring Transaction Management Strategy for an MDB.	For message listener methods: `Required` or `NotSupported`. For timeout callback methods: `Required`, `RequiresNew` or `NotSupported`. The corresponding annotation values are `REQUIRED`, `REQUIRES_NEW` and `NOT_SUPPORTED`, subject to the same method restrictions.	`Required`

Copyright © 2007, 2026, Oracle and/or its affiliates.

Table 11-1 (Cont.) Deployment Elements and Annotations for MDBs

Element	Configuration Property	Description	Allowable Values	Default
`trans-timeout-seconds` (`weblogic-ejb-jar.xml`)	`value` member of `@weblogic.javaee.TransactionTimeoutSeconds`. Example on the MDB class: `@weblogic.javaee.TransactionTimeoutSeconds(60)`.	The maximum duration for an EJB's container-initiated transactions, in seconds, after which the transaction is rolled back. See Configuring Transaction Management Strategy for an MDB.	Non-negative integer, `0` through 2147483647.	When no MDB timeout is supplied, or its effective value is `0`, the domain JTA timeout is used; the domain default is 30 seconds. A `@TransactionTimeoutSeconds` annotation with no argument specifies 30 seconds, unless overridden by the deployment descriptor.
`use81-style-polling` (`weblogic-ejb-jar.xml`)	`use81StylePolling`	Enables backwards compatibility for Oracle WebLogic Server version 8.1-style polling.	• `True` • `False`	False
`activation-config-property` (`ejb-jar.xml`)	`aqMDBReceiveNoWait`	If AQ JMS is the JMS provider for the MDB and this property is set to true, synchronous polling uses the `MessageConsumer.receiveNoWait()` method to optimize AQ JMS dequeue performance.	• `True` • `False`	False
`activation-config-property` (`ejb-jar.xml`)	`mdbDestinationPollIntervalMillis`	When the JMS destination has no messages and synchronous polling is used for MDB, this property sets the message polling interval in milliseconds. When MDB interoperates with AQ JMS and `minimizeAQSessions` is set to true, you can avoid database connection usage bursts by setting this property to a value that is greater than 5000. This configuration randomly starts message polling when multiple MDBs are started.	> or = 0 (Integer)	0

Copyright © 2007, 2026, Oracle and/or its affiliates.

Table 11-1 (Cont.) Deployment Elements and Annotations for MDBs

Element	Configuration Property	Description	Allowable Values	Default
`activation-config-property` (`ejb-jar.xml`)	`minimizeAQSessions`	This property is only used when MDB interoperates with AQ JMS. When this property is set to true, it reduces database resource usage by dynamically closing AQ JMS sessions. Dynamic session closing does not apply to non-durable topic subscriptions.	• `True` • `False`	False

Copyright © 2007, 2026, Oracle and/or its affiliates.

A

Topic Deployment Scenarios

Examine the MDB deployment actions and typical scenarios for various topic MDB configurations.
The actions include where and how many MDB free pools are created, where and how many subscriptions are created, and how the subscribers work together to achieve a given messaging consumption pattern.

These sections do not cover details about legacy behavior which occurs when the `topicMessagesDistributionMode` is set to `Compatibility`, when the topics are foreign (non-WebLogic) topics, or when the topics are WebLogic JMS topics from Oracle WebLogic Server releases prior to 10.3.4.

For help determining the right scenario (permutation) for your application, including suggested settings, see Configuring and Deploying MDBs Using JMS Topics .

This appendix includes the following sections:

- How Configuration Permutations Determine Deployment Actions
- Typical Scenarios

How Configuration Permutations Determine Deployment Actions

Determine how WebLogic MDBs that consume from WebLogic JMS topics from Oracle WebLogic Server 10.3.4 or later create instances of MDB free pools, subscription naming, subscription locations, and how messages are distributed to those MDB pool instances.
The following settings determine these deployment actions:

- The topic location (in the same cluster or server as the MDB deployment or on a remote cluster or server).
- The topic type (singleton WebLogic topic, `Replicated` or `Partitioned` distributed topic).
- The `subscriptionDurability` setting.
- The `topicMessagesDistributionMode` and `distributedDestinationConnection` settings.

Table A-1 describes possible configuration permutations and corresponding deployment actions. The first two columns describe the configuration permutations, and the last two columns describe the resulting deployment. The columns are as follows:

- **`topicMessagesDistributionMode`** -- The value of the `topicMessagesDistributionMode` configuration option, that is, `One-Copy-Per-Server` or `One-Copy-Per-Application`. The legacy `Compatibility` mode is not covered in this table.
- **Topic Type Permutation** -- Options include the following:
  - **Local** or **Remote** -- Whether the topic is deployed to the same cluster or server as the MDB (Local) or to a different cluster or server (Remote).
  - **PDT**, **RDT**, or **Singleton WebLogic JMS topic** -- The type of topic: partitioned distributed topic (PDT), replicated distributed topic (RDT), or singleton WebLogic JMS topic.

Copyright © 2007, 2026, Oracle and/or its affiliates.

  - **EveryMember** or **LocalOnly** -- The value of `distributedDestinationConnection`. Specifies whether the MDB that accesses a `Local` distributed topic in the same cluster consumes from all distributed topic members or only from those local to the current server. If neither `EveryMember` nor `LocalOnly` is specified, the permutation applies regardless of how `distributedDestinationConnection` is set.

  For example, the topic type permutation "Local RDT LocalOnly" means "An MDB is deployed to the same cluster (Local) as the replicated topic (RDT), and the MDB is configured to consume only from members of the topic on the same Oracle WebLogic Server as the MDB (`LocalOnly`)."

- **Each Server Subscribes to...** -- The number of MDB pools an Oracle WebLogic Server instance creates, and the members of the distributed topic to which the MDB instances subscribe. For example,
  - "**Each server subscribes to** ... **All members**" means "the container creates one local MDB pool for each member of the distributed topic."
  - "**Each server subscribes to** ... **All local members**" means "the container creates one MDB pool for each of the members that are running on the same server, and each MDB pool subscribes to one of those members."
- **MDB Pools Per Server** -- The number of MDB deployment instances on each server in the cluster (and thereby the number of connections to the distributed topic members). ***M*** = the number of distributed topic members (*M*=1 for standalone topics).

Table A-1 Configuration Permutations and Their Resulting Deployment Actions

MDB Configuration		Deployment Actions	
topicMessagesDistributionMode	**Topic Type Permutation**	**Each Server Subscribes to...**	**MDB Pools per Server**
`One-Copy-Per-Server`	• RDT Local LocalOnly[1]	One of the local members	One
`One-Copy-Per-Server`	• RDT Remote (Non-durable only)[2]	One of the remote members	One
`One-Copy-Per-Server`	• PDT Local EveryMember[3] • PDT Remote[2] • RDT Local EveryMember • RDT Remote (durable subscriptions only)[2] • Singleton WebLogic JMS (`M`=1)	All members	*M*
`One-Copy-Per-Application`	• PDT Local LocalOnly • RDT Local LocalOnly	All local members	One per local member
`One-Copy-Per-Application`	• PDT Local EveryMember • PDT Remote[2] • RDT Local EveryMember • RDT Remote[2] • Singleton WebLogic JMS (M=1)	All members	*M*

[1] The "`One-Copy-Per-Server`, RDT, Local, LocalOnly" permutation *is not supported* for durable subscription cases in some configuration topologies (See details in Scenario 1: Replicated DT, One Copy Per Server, Local Deployment, Local Only Consumption.)

[2] For remote distributed topics, Oracle WebLogic Server always creates subscriptions to every topic member except for non-durable subscriptions in the "`One-Copy-Per-Server`, Replicated Distributed Topic, Remote" permutation. In that case, only one subscription to one of the remote members is created. (See Scenario 3: Replicated DT, One Copy Per Server, Remote Deployment.)

Copyright © 2007, 2026, Oracle and/or its affiliates.

[3] The `LocalOnly` setting is always automatically replaced with `EveryMember` in the "One-Copy-Per-Server, Partitioned Distributed Topic, Local" permutation. (See Scenario 7: Partitioned DT, One Copy Per Server, Local Deployment, Local Only Consumption.)

Typical Scenarios

Examine the Standalone Non-distributed, Replicated Distributed and Partitioned Distributed scenarios of an MDB application.
The following sections show the possible deployment scenarios of an MDB application:

- Standalone (Non-distributed) Topic Scenarios
- Replicated Distributed Topic Scenarios
- Partitioned Distributed Topic Scenarios

Images and labels used in the figures presented in the scenarios are explained in Table A-2:

Table A-2 Explanation of Images and Text Used in Scenarios

Image or Text	Explanation
	Messages published to a distributed topic.
	Messages are duplicated, and copies are forwarded to other members of the topic. This indicates that the topics are replicated distributed topics.
DT Member *n*	Member of a distributed topic.
MDB Pool	An MDB free bean pool.
	A subscription. The MDB on one end of the arrow listens for and consumes messages from the topic on the other end of the arrow.
Subscription 1, 2, and 3	Non-shared subscriptions. Subscription 1 is Managed Server 1's subscription, Subscription 2 is Managed Server 2's subscription, and so on.

Standalone (Non-distributed) Topic Scenarios

Standalone topic scenarios are as follows.

One-Copy-Per-Server

On each Oracle WebLogic Server instance that hosts the MDB application, an MDB pool is created for the topic, whether the topic is running in the same cluster or in a different cluster.

Copyright © 2007, 2026, Oracle and/or its affiliates.

For an MDB cluster of **N** nodes, **N** MDB pools are created. Each MDB pool creates an individual subscription on the topic, and subscribers from different MDB pools do not share the same subscription.

One-Copy-Per-Application

On each Oracle WebLogic Server instance that hosts the MDB application, an MDB pool is created for the topic, whether the topic is running in the same cluster or in a different cluster. For an MDB cluster of **N** nodes, **N** MDB pools are created. All subscribers created by the MDB pools of the same MDB application share the same subscription.

Replicated Distributed Topic Scenarios

With replicated distributed topics, all physical topic members receive each message sent. When a message arrives at one of the physical topic members, a copy of the message is automatically internally forwarded to the other members of the topic.

The following are the possible deployment scenarios for a replicated distributed topic:

- Scenario 1: Replicated DT, One Copy Per Server, Local Deployment, Local Only Consumption
- Scenario 2: Replicated DT, One Copy Per Server, Local Deployment, Every Member Consumption
- Scenario 3: Replicated DT, One Copy Per Server, Remote Deployment
- Scenario 4: Replicated DT, One Copy Per Application, Local Deployment, Local Only Consumption
- Scenario 5: Replicated DT, One Copy Per Application, Local Deployment, Every Member Consumption
- Scenario 6: Replicated DT, One Copy Per Application, Remote Deployment

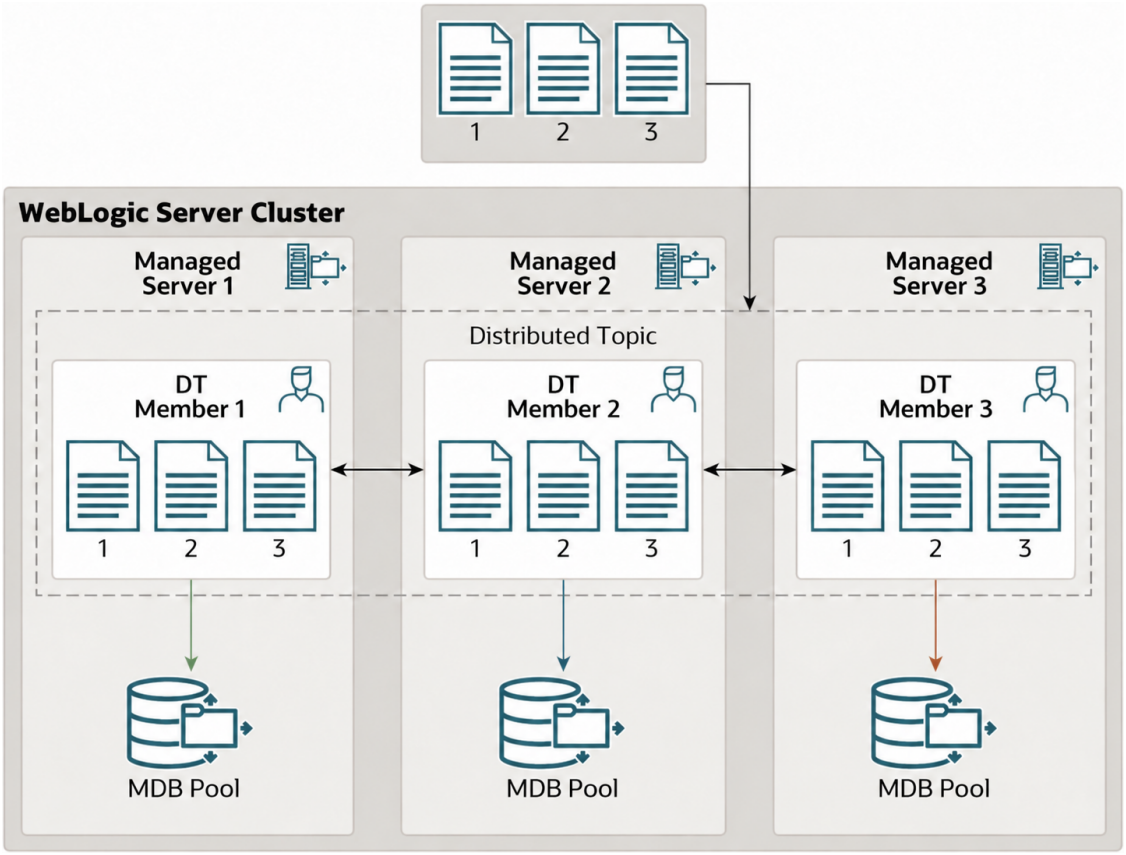
Scenario 1: Replicated DT, One Copy Per Server, Local Deployment, Local Only Consumption

Figure A-1 shows the following configuration:

- Replicated distributed topic
- `topicMessagesDistributionMode` = `One-Copy-Per-Server`.
- The MDB and the topic are deployed in the same (local) cluster.
- `distributedDestinationConnection` = `LocalOnly`.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Figure A-1 Scenario 1: Replicated DT, One Copy Per Server, Local Deployment, Local Only Consumption



In this scenario:

- Copies of messages are forwarded to other servers in the cluster by the RDT.
- One MDB pool is created on each server in the local cluster.
- Each MDB pool listens to one of the distributed topic members on the same server.

This approach can reduce network traffic for MDB consumption because each MDB pool consumes from a topic member on the same server. RDT replication between servers still requires network communication. For some workloads, the EveryMember alternative may perform better; see Handling Uneven Message Loads and/or Message Processing Delays.

This scenario does not work correctly for durable subscriptions when there are multiple members on the same server, when there are no members on any of the local servers that host the MDB application, or when JMS service migration (auto or manual) is involved.

Scenario 2: Replicated DT, One Copy Per Server, Local Deployment, Every Member Consumption

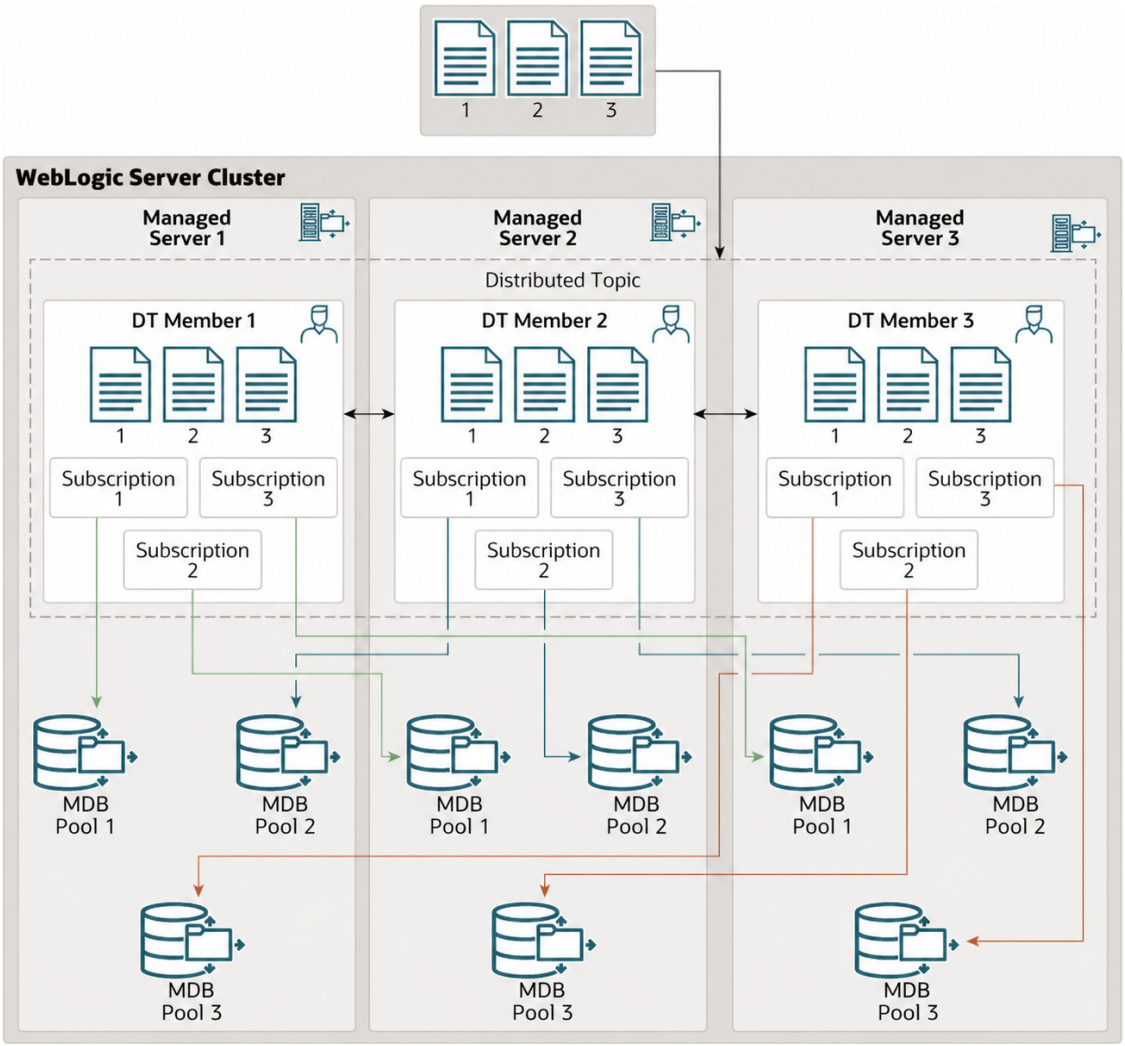
Figure A-2 shows the following configuration:

- Replicated distributed topic
- `topicMessagesDistributionMode` = `One-Copy-Per-Server`.
- The MDB and the topic are deployed in the same (local) cluster.

Copyright © 2007, 2026, Oracle and/or its affiliates.

- `distributedDestinationConnection = EveryMember`.

Figure A-2 Scenario 2: Replicated DT, One Copy Per Server, Local Deployment, Every Member Consumption



In this scenario:

- Copies of messages are forwarded to other servers in the cluster by the RDT, but these copies are filtered out (ignored) by the MDB subscriptions.
- An MDB pool is created for each distributed topic member on each server in the local cluster.
- Each MDB pool listens to one of the distributed topic members in the cluster.
- Each Oracle WebLogic Server instance that hosts the MDB application listens to all members of the distributed topic.
- Each server's subscribers on the same member of the DT have their own independent subscriptions. In other words, subscribers from different servers to the same member do not share any subscriptions.

Copyright © 2007, 2026, Oracle and/or its affiliates.

This configuration yields high flexibility and is good for an application where an RDT is required, but it cannot be guaranteed that there will be exactly one member per server, for example due to migration.

Compared with Scenario 1, this configuration can add network traffic for MDB consumption. Its performance depends on the workload and deployment. For configurations with exactly one topic member on each managed server and no JMS service migration requirement, consider Scenario 1 as an alternative. See Handling Uneven Message Loads and/or Message Processing Delays for the trade-offs.

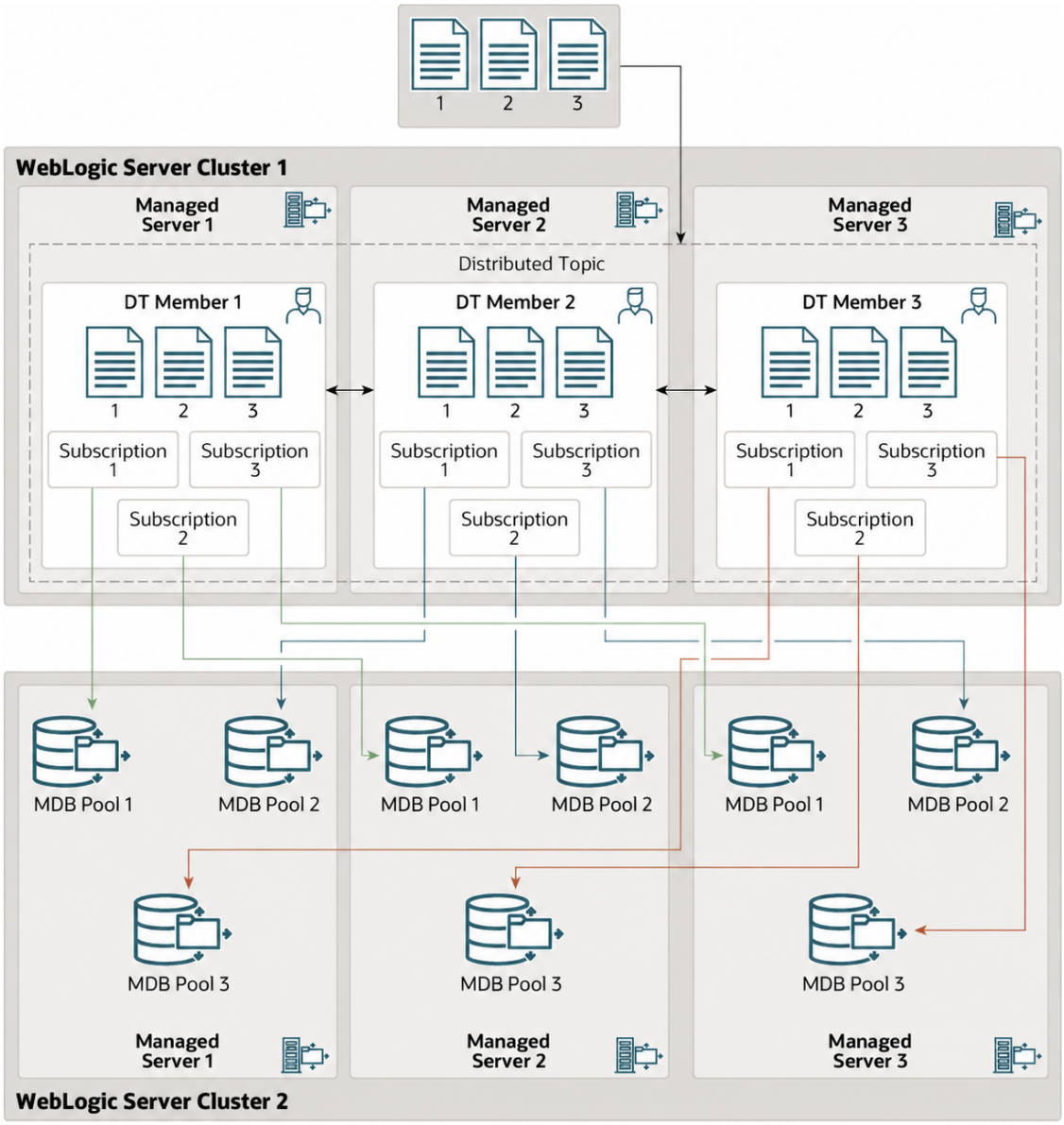
Scenario 3: Replicated DT, One Copy Per Server, Remote Deployment

Figure A-3 shows the following configuration:

- Replicated distributed topic
- Durable subscription
- `topicMessagesDistributionMode` = `One-Copy-Per-Server`.
- The MDB and the topic are deployed in different (remote) clusters.
- `distributedDestinationConnection` ignored for remote deployments.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Figure A-3 Scenario 3: Replicated DT, One Copy Per Server, Remote Deployment



In this scenario:

- Copies of messages are forwarded to other servers in the cluster by the RDT, but these copies are filtered out (ignored) by the MDB subscriptions.
- An MDB pool for each distributed topic member is created on each server in the remote cluster.
- Each Oracle WebLogic Server instance that hosts the MDB application listens to all members of the distributed topic (one local pool for each remote member).
- Each server's subscribers on the same member of the DT have their own independent subscription. In other words, subscribers from different servers to the same member do not share any subscriptions.

Copyright © 2007, 2026, Oracle and/or its affiliates.

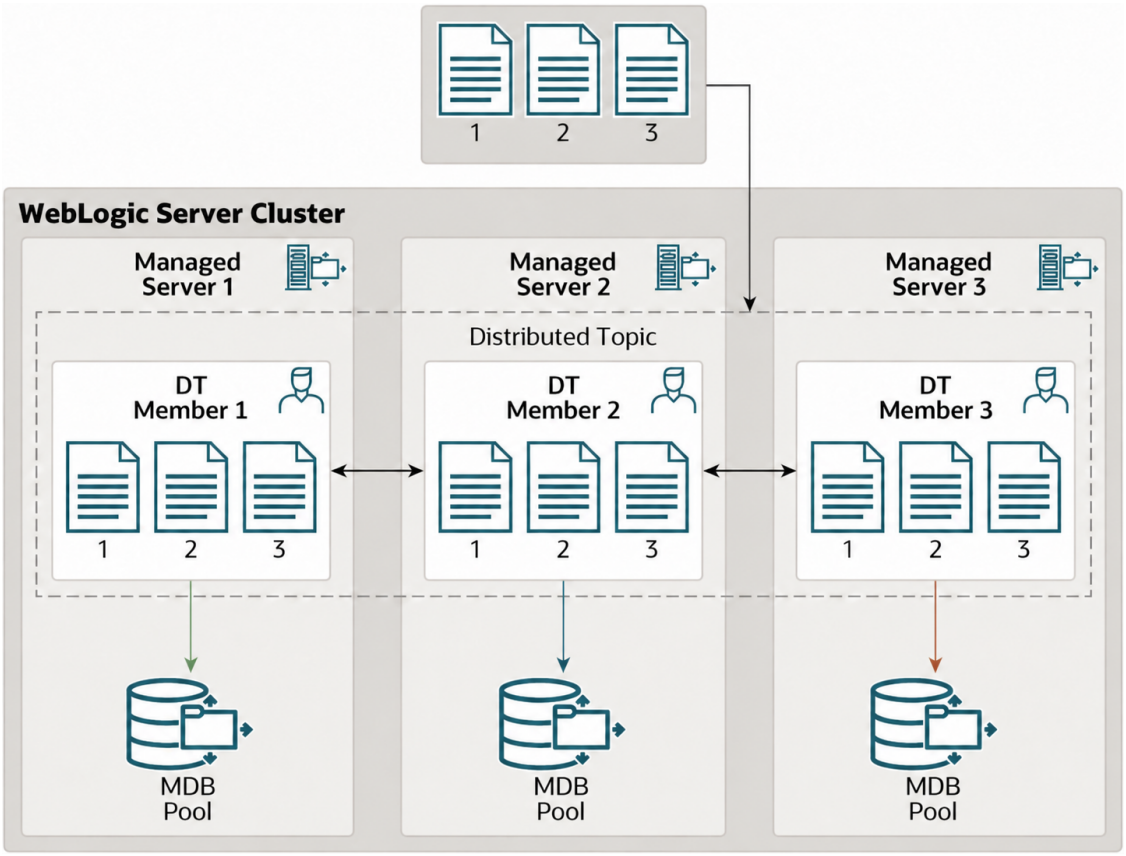
Note that this is the behavior for durable cases. For non-durable cases, each Oracle WebLogic Server instance creates a single MDB pool which connects to one of the members (any member) as an optimization.

Scenario 4: Replicated DT, One Copy Per Application, Local Deployment, Local Only Consumption

Figure A-4 shows the following configuration:

- Replicated distributed topic
- `topicMessagesDistributionMode` = `One-Copy-Per-Application`.
- The MDB and the topic are deployed in the same (local) cluster.
- `distributedDestinationConnection` = `LocalOnly`.

Figure A-4 Scenario 4: Replicated DT, One Copy Per Application, Local Deployment, Local Only Consumption



In this scenario:

- Copies of messages are forwarded to other servers in the cluster by the RDT, but these copies are filtered out (ignored) by the MDB subscriptions.
- One MDB pool is created on each server in the local cluster for each local member (Figure A-4 shows a configuration where each Oracle WebLogic Server instance hosts only one member. When there are multiple members on the same local Oracle WebLogic Server instance, multiple MDB pools are created on the server).

Copyright © 2007, 2026, Oracle and/or its affiliates.

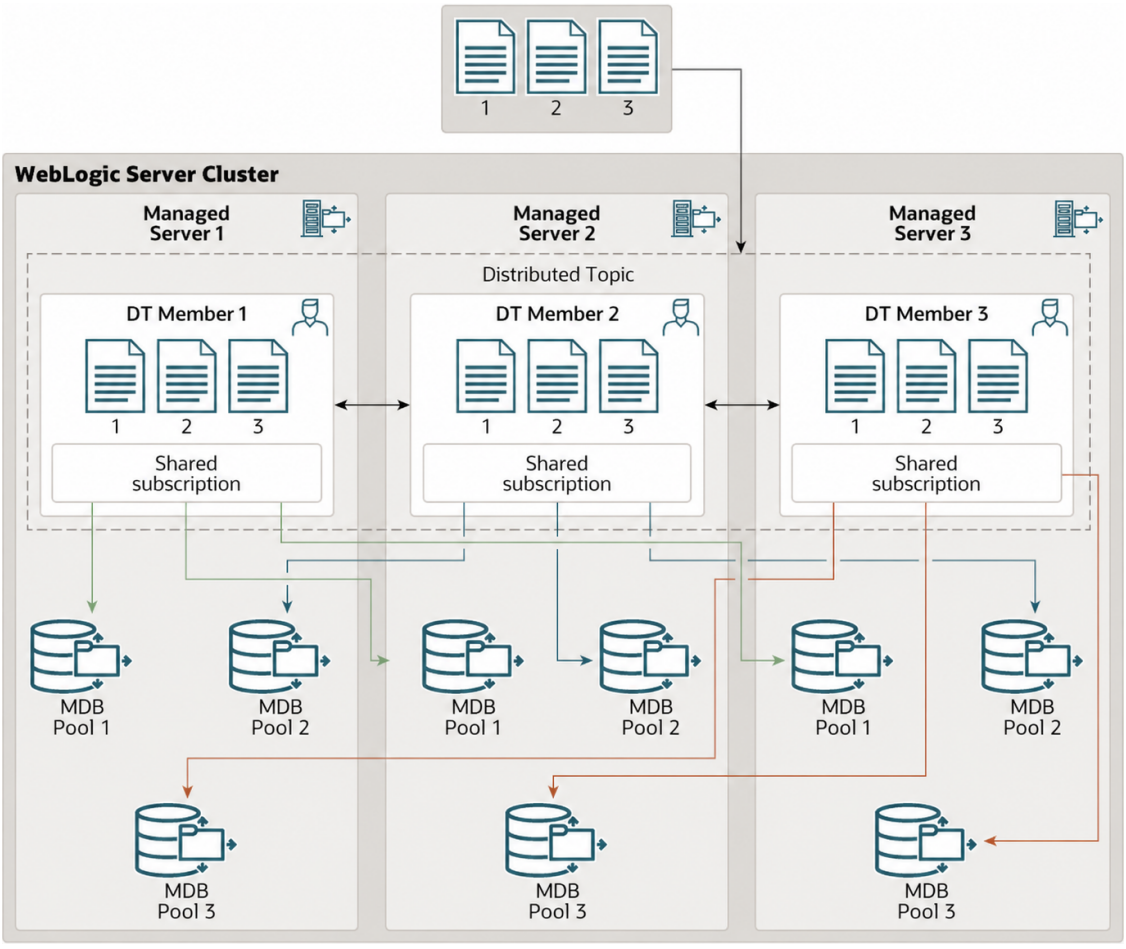
- A message is given to only one MDB pool.
- All subscribers on the same member share the same subscription.

Scenario 5: Replicated DT, One Copy Per Application, Local Deployment, Every Member Consumption

Figure A-5 shows the following configuration:

- Replicated distributed topic
- `topicMessagesDistributionMode` = `One-Copy-Per-Application`.
- The MDB and the topic are deployed in the same (local) cluster.
- `distributedDestinationConnection` = `EveryMember`.

Figure A-5 Scenario 5: Replicated DT, One Copy Per Application, Local Deployment, Every Member Consumption



In this scenario:

- Copies of messages are forwarded to other servers in the cluster by the RDT, but these copies are filtered out (ignored) by the MDB subscriptions.

Copyright © 2007, 2026, Oracle and/or its affiliates.

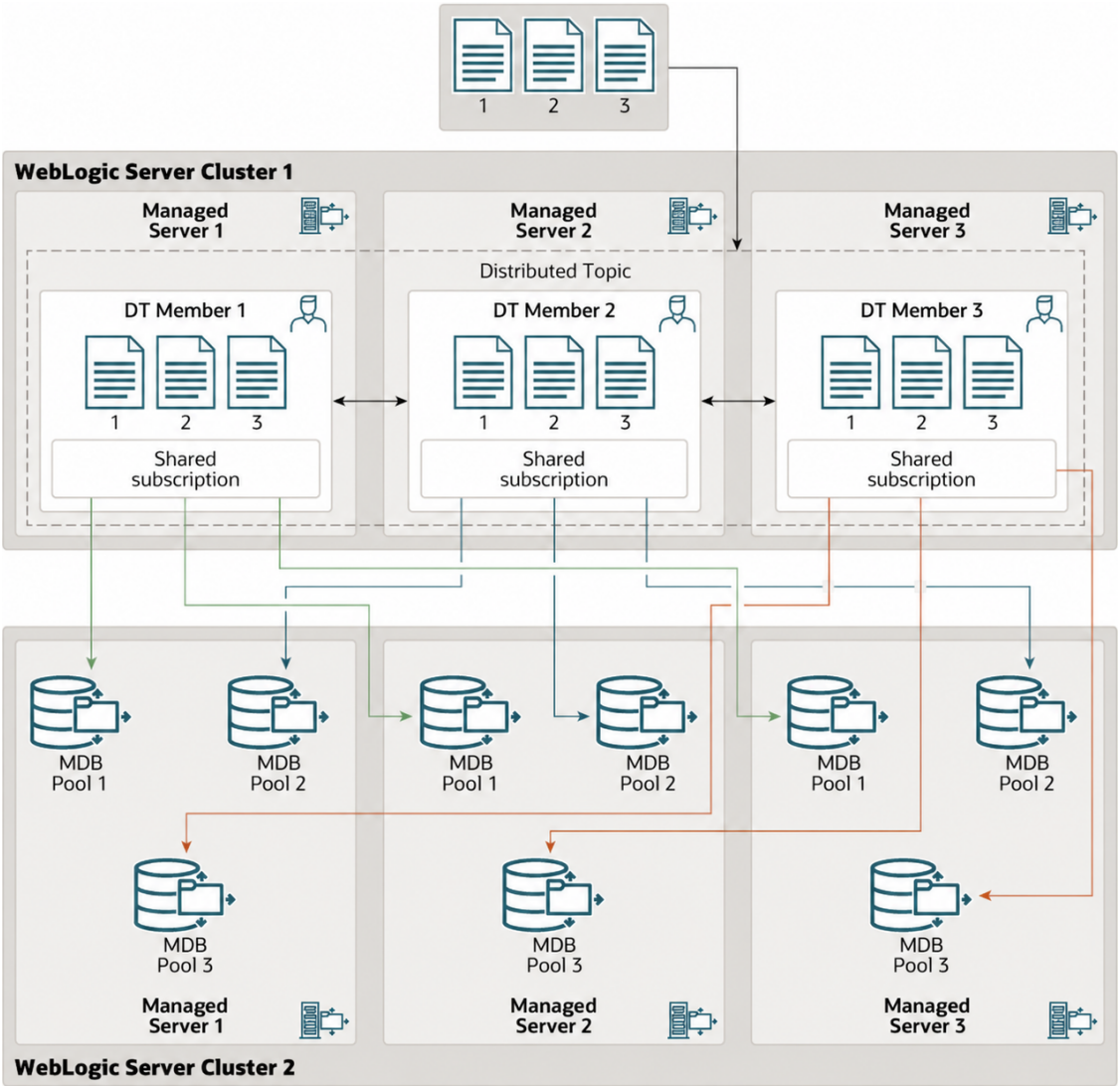
- One MDB pool is created on each server in the local cluster for each member.
- A message is given to only one MDB pool.
- All subscribers on the same member share the same subscription.

Scenario 6: Replicated DT, One Copy Per Application, Remote Deployment

Figure A-6 shows the following configuration:

- Replicated distributed topic
- `topicMessagesDistributionMode` = `One-Copy-Per-Application`.
- The MDB and the topic are deployed in different (remote) clusters.
- `distributedDestinationConnection` ignored for remote deployments.

Figure A-6 Scenario 6: Replicated DT, One Copy Per Application, Remote Deployment



In this scenario:

Copyright © 2007, 2026, Oracle and/or its affiliates.

- Copies of messages are forwarded to other servers in the cluster by the RDT, but these copies are filtered out (ignored) by the MDB subscriptions.
- One MDB pool is created on each server in the local cluster for each member in the remote cluster.
- A message is given to only one MDB pool.
- All subscribers on the same member share the same subscription.

Partitioned Distributed Topic Scenarios

With partitioned topics:

- The distributed topic member receiving the message is the only member that is aware of the message. The message is not forwarded to other members, and subscribers on other members do not get a copy of the message.
- Incoming messages can be load balanced among PDT members. Producer-side routing is influenced by the Load Balancing Enabled and Server Affinity Enabled settings on the connection factory used by the producer. See Load Balancing Partitioned Distributed Topics in *Administering JMS Resources for Oracle WebLogic Server*.

The following are the possible deployment scenarios for a partitioned distributed topic:

- Scenario 7: Partitioned DT, One Copy Per Server, Local Deployment, Local Only Consumption
- Scenario 8: Partitioned DT, One Copy Per Server, Local Deployment, Every Member Consumption
- Scenario 9: Partitioned DT, One Copy Per Server, Remote Deployment
- Scenario 10: Partitioned DT, One Copy Per Application, Local Deployment, Local Only Consumption
- Scenario 11: Partitioned DT, One Copy Per Application, Local Deployment, Every Member Consumption
- Scenario 12: Partitioned DT, One Copy Per Application, Remote Deployment

Scenario 7: Partitioned DT, One Copy Per Server, Local Deployment, Local Only Consumption

The setting of `distributedDestinationConnection` is ignored for this scenario and a warning message is logged. The setting is forced to `EveryMember` instead. The behavior becomes the same as the "`EveryMember`" case (see Scenario 8: Partitioned DT, One Copy Per Server, Local Deployment, Every Member Consumption).

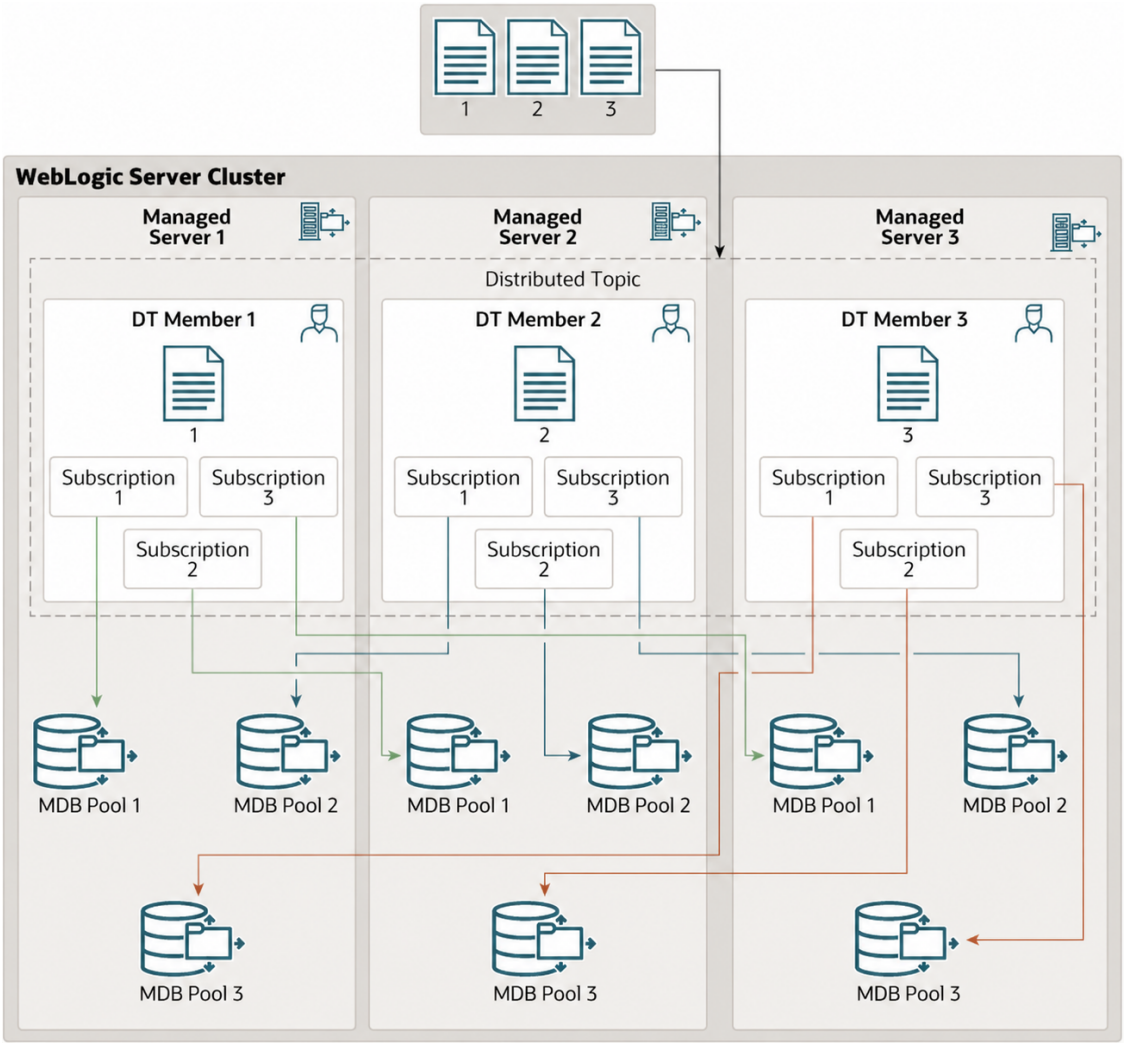
Scenario 8: Partitioned DT, One Copy Per Server, Local Deployment, Every Member Consumption

Figure A-7 shows the following configuration:

- Partitioned distributed topic
- `topicMessagesDistributionMode` = `One-Copy-Per-Server`.
- The MDB and the topic are deployed in the same (local) cluster.
- `distributedDestinationConnection` = `EveryMember`.

Copyright © 2007, 2026, Oracle and/or its affiliates.

Figure A-7 Scenario 8: Partitioned DT, One Copy Per Server, Local Deployment, Every Member Consumption



In this scenario:

- Messages are distributed individually to the distributed topic members. Messages are not duplicated or copied to other members in the cluster.
- An MDB pool is created for each distributed topic member on each server in the local cluster.
- Each server's subscribers on the same member of the DT have their own independent subscription. In other words, subscribers from a particular server to the same member do not share any subscriptions with subscribers from another server.

Scenario 9: Partitioned DT, One Copy Per Server, Remote Deployment

The details of this scenario are the same as the previous one except that the MDB deployment and the PDT are in different clusters.

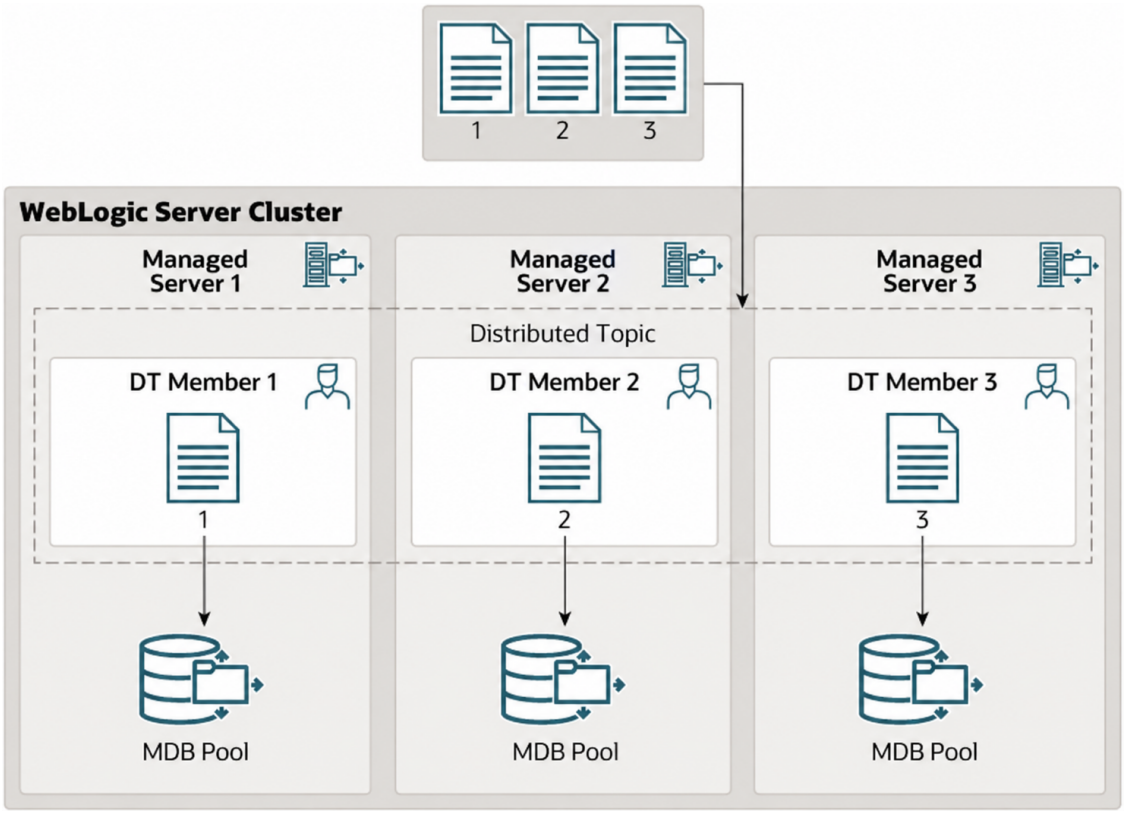
Copyright © 2007, 2026, Oracle and/or its affiliates.

Scenario 10: Partitioned DT, One Copy Per Application, Local Deployment, Local Only Consumption

Figure A-8 shows the following configuration:

- Partitioned distributed topic
- `topicMessagesDistributionMode` = `One-Copy-Per-Application`.
- The MDB and the topic are deployed in the same (local) cluster.
- `distributedDestinationConnection` = `LocalOnly`.

Figure A-8 Scenario 10: Partitioned DT, One Copy Per Application, Local Deployment, LocalOnly Consumption



In this scenario:

- Messages are distributed individually to the distributed topic members. Messages are not duplicated or copied to other members in the cluster.
- One MDB pool is created on each server in the local cluster for each local member (Figure A-8 shows a configuration where each Oracle WebLogic Server hosts only one member. When there are multiple members on the same local Oracle WebLogic Server, multiple MDB pools are created on each Oracle WebLogic Server instance).

This scenario is the recommended configuration for `One-Copy-Per-Application` and local PDT for high performance. However for some use cases, the `EveryMember` alternative may work better, based on the trade-offs discussed in Handling Uneven Message Loads and/or Message Processing Delays.

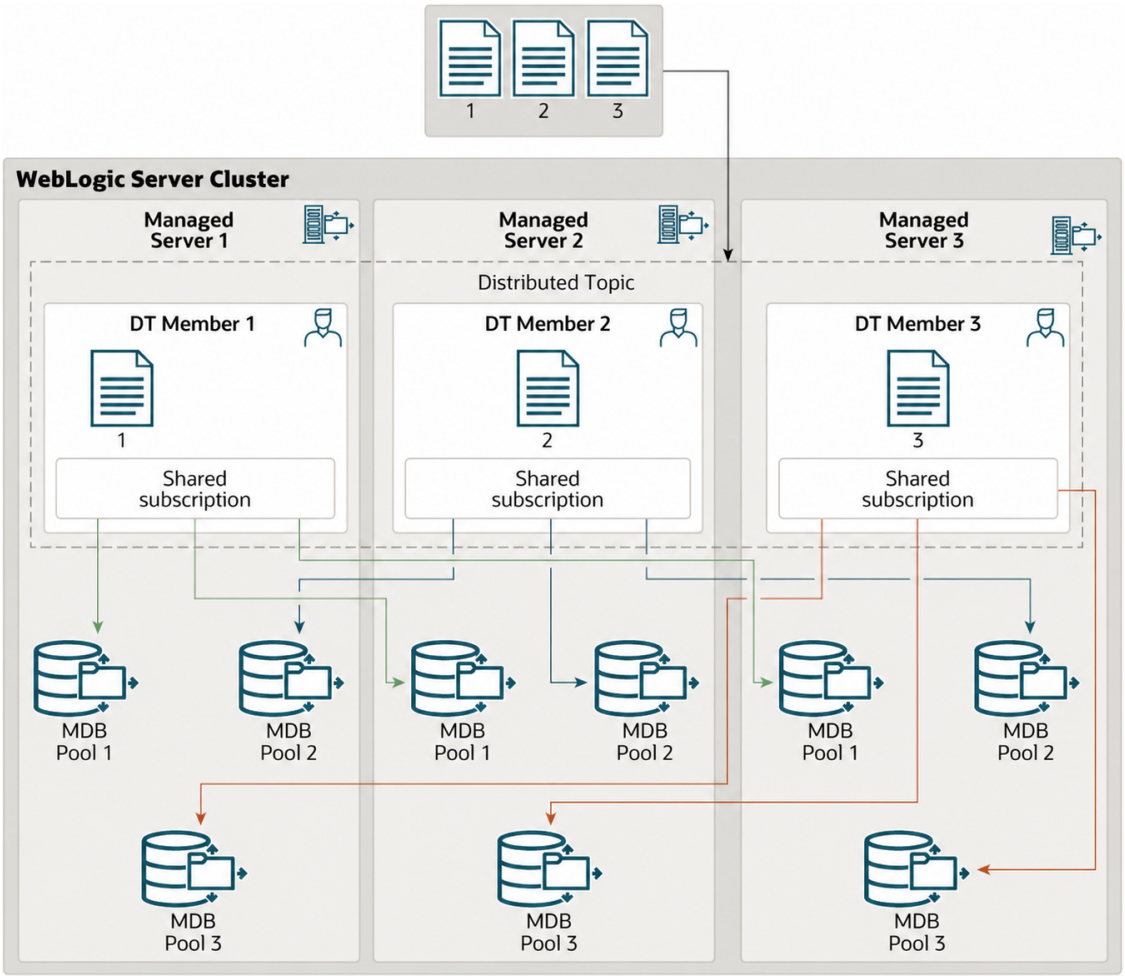
Copyright © 2007, 2026, Oracle and/or its affiliates.

Scenario 11: Partitioned DT, One Copy Per Application, Local Deployment, Every Member Consumption

Figure A-9 shows the following configuration:

- Partitioned distributed topic
- `topicMessagesDistributionMode` = `One-Copy-Per-Application`.
- The MDB and the topic are deployed in the same (local) cluster.
- `distributedDestinationConnection` = `EveryMember`.

Figure A-9 Scenario 11: Partitioned DT, One Copy Per Application, Local Deployment, Every Member Consumption



For a partitioned distributed topic, one copy per application, local deployment, it is better to use `LocalOnly` consumption for most use cases, as shown in Scenario 10: Partitioned DT, One Copy Per Application, Local Deployment, Local Only Consumption. However the trade-offs discussed in Handling Uneven Message Loads and/or Message Processing Delays, apply here.

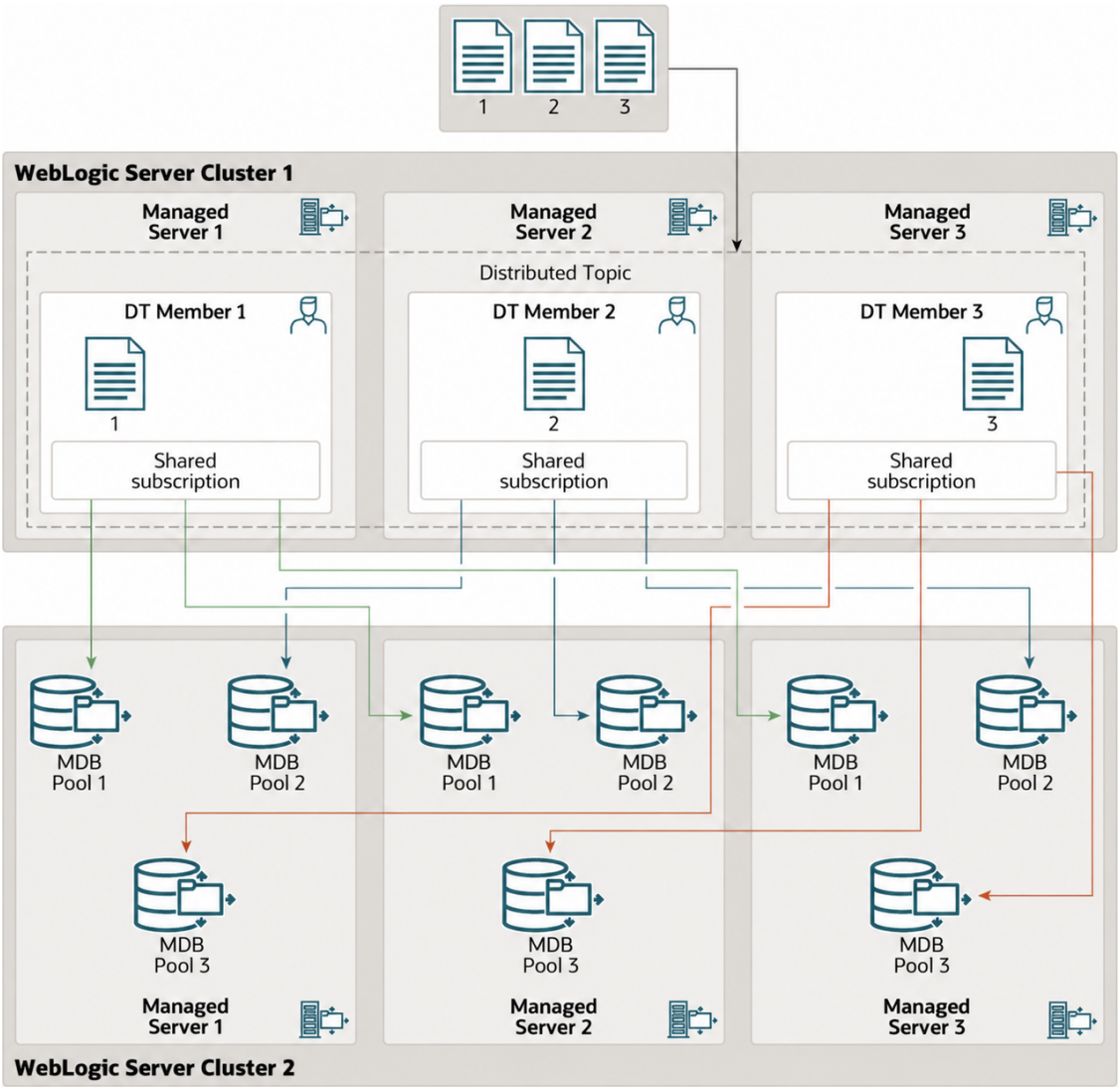
Copyright © 2007, 2026, Oracle and/or its affiliates.

Scenario 12: Partitioned DT, One Copy Per Application, Remote Deployment

Figure A-10 shows the following configuration:

- Partitioned distributed topic
- `topicMessagesDistributionMode` = `One-Copy-Per-Application`.
- The MDB and the topic are deployed in different (remote) clusters.
- `distributedDestinationConnection` ignored for remote deployments.

Figure A-10 Scenario 12: Partitioned DT, One Copy Per Application, Remote Deployment



In this scenario:

- Messages are distributed individually to the distributed topic members. Messages are not duplicated or copied to other members in the cluster.

Copyright © 2007, 2026, Oracle and/or its affiliates.

- Subscriptions are created on all the distributed topic members automatically and dynamically.
- A message is given to only one MDB pool.
- All subscribers from the same MDB application on the same member share the same subscription.

Copyright © 2007, 2026, Oracle and/or its affiliates.

B

Topic Subscription Identifiers

Examine how the topic subscription identifiers are generated.
For MDBs using WebLogic Server's built-in JMS integration, subscription selection uses the following information, as applicable:

1. The topic with which the subscription is associated
2. The connection "Client ID" string that is specified for the connection that is used to access the subscription
3. For a durable subscription, the subscription name. The MDB container uses the configured `subscriptionName` value, if provided; otherwise, it derives the name as shown in Table B-1. It supplies that name when creating the durable subscriber.
4. The "Client ID Policy" option, by which a subscription is also identified in WebLogic JMS. If two WebLogic JMS subscription references on the same physical topic have the same client ID and subscription name, then the references resolve to:
   - A single subscription, if the client ID policies are also the same
   - Two different subscriptions, if the client ID policies are different

The MDB container uses the configured destination and determines the applicable client ID, durable subscription name, and Client ID Policy from the following settings:

- `ejb-name`
- `jms-client-id`
- `topicMessagesDistributionMode`
- `subscriptionName` configuration property
- `distributedDestinationConnection`
- `generate-unique-jms-client-id`
- `subscriptionDurability`
- Other elements of the MDB deployment and JMS configurations

Configure `generate-unique-jms-client-id` under `message-driven-descriptor` in `weblogic-ejb-jar.xml`, or use the `generateUniqueID` member of `@weblogic.javaee.JMSClientID`.

The `generate-unique-jms-client-id` setting affects client-ID generation only in `Compatibility` mode. The `distributedDestinationConnection` setting controls local distributed-topic member selection in the One-Copy modes and can also affect generated client IDs in `Compatibility` mode. The `subscriptionDurability` setting applies in all three modes.

Table B-1 shows the client-ID expressions generated from the MDB's configuration. For a durable MDB in `Compatibility` mode, the effective connection client ID is the ID already assigned by the connection factory, if present; otherwise, the container sets the generated client ID. Unless `subscriptionName` is configured, the durable subscription name is derived from the generated client-ID expression in the corresponding row, even when the connection uses a different preconfigured client ID. For the One-Copy modes, leave the connection

Copyright © 2007, 2026, Oracle and/or its affiliates.

factory's client ID unset and configure the client-ID base on the MDB; the container must set the connection's client ID itself.

Table B-1 summarizes how the settings are used to generate subscription IDs:

Table B-1 How Subscription IDs are Generated

Setting	Generated Client ID	Subscription Name for the Durable Subscription Case	Client ID Policy for WebLogic Topics
`topicMessagesDistributionMode` = `One-Copy-Per-Application`	`jmsClientIDBase`	`subscriptionName` property, if used, or `ejb-name`	Unrestricted
`topicMessagesDistributionMode` = `One-Copy-Per-Server`	`jmsClientIDBase` + `"_"` + `currentDomainName` + `"_"` + `currentServerName`	`subscriptionName` property, if used, or `ejb-name`	Unrestricted
`topicMessagesDistributionMode` = `Compatibility` `generate-unique-jms-client-id` = `true` `distributedDestinationConnection` = `LocalOnly` `subscriptionDurability` = `Durable` Foot 1	`jmsClientIDBase` + `"_"` + `currentDomainName` + `"_"` + `uniqueKey`	`subscriptionName`, if configured; otherwise, the generated client ID shown in this row.	Connection factory's Client ID Policy; Restricted by default.
Same as previous row, except: `distributedDestinationConnection` = `EveryMember`	`jmsClientIDBase` + `"_"` + `currentDomainName` + `"_"` + `uniqueKey` + `"_"` + `DDMemberName`	`subscriptionName`, if configured; otherwise, the generated client ID shown in this row.	Connection factory's Client ID Policy; Restricted by default.
`topicMessagesDistributionMode` = `Compatibility` `generate-unique-jms-client-id` = `false` `subscriptionDurability` = `Durable` Foot 1	`jmsClientIDBase`	`subscriptionName`, if configured; otherwise, the generated client ID shown in this row.	Connection factory's Client ID Policy; Restricted by default.

Foot 1. For non-durable MDBs in Compatibility mode, the container does not assign a client ID or a durable subscription name. A client ID already supplied by the connection factory remains in effect. For WebLogic JMS, the connection factory's Client ID Policy applies and defaults to Restricted.

Key:

- `jms-client-id` — Optional MDB client-ID configuration in `weblogic-ejb-jar.xml`. The MDB client-ID base can also be configured using an activation property or annotation; see Deployment Elements and Annotations for MDBs. A client ID configured on the JMS connection factory is a separate connection setting.
- `subscriptionName` — an optional setting to establish the durable subscription name. If specified, this setting takes precedence over any automatically generated name.

Copyright © 2007, 2026, Oracle and/or its affiliates.

- `ejb-name` = the name of the EJB
- `jmsClientIDBase` — The client-ID base obtained from the MDB's configuration, or `ejb-name` if no MDB client ID is configured. A client ID configured on the JMS connection factory does not supply this base.
- `currentDomainName` — the name of the WebLogic domain that runs the MDB
- `currentServerName` — the name of the Oracle WebLogic Server that the MDB is running on
- `uniqueKey` — A generated value combining MDB deployment information with either `currentServerName` or the name of the migratable target associated with that MDB connection.
- `DDMemberName` — the name of a distributed destination member; or, alternatively, the destination name if the topic is a singleton or a distributed destination in releases of Oracle WebLogic Server prior to 10.3.4.

Client ID uniqueness is enforced as follows:

- *For foreign (non-WebLogic) JMS vendors:* Some JMS vendors prevent more than one connection from specifying the same connection Client ID. (An exception is thrown on an attempt to create the second connection.) This limitation in turn can prevent more than one free pool from using the same Client ID, because each free pool creates a single JMS connection with potentially the same Client ID as other free pool connections. After a first free pool instance of the MDB starts on a server instance in the cluster, an additional instance of the EJB can deploy successfully on another clustered server; but when the MDB attempts to create a JMS connection, a Client ID conflict is detected and that instance of the MDB fails to fully connect to JMS.
- *For WebLogic JMS:* For WebLogic JMS in releases of Oracle WebLogic Server prior to 10.3.4, JMS connections were restricted so only one connection with the same Client ID could exist in the scope of a cluster. However, for Oracle WebLogic Server 10.3.4 and later, WebLogic JMS connection factories or connections can optionally set a Client ID Policy to control this restriction. With a Client ID Policy of RESTRICTED, the pre-10.3.4 behavior remains in effect, while with a Client ID Policy of UNRESTRICTED, this limitation is lifted. See *Developing JMS Applications for Oracle WebLogic Server*. Unrestricted client IDs make it possible for multiple WebLogic subscriber connections and subscriptions to share the same client ID. Both `One-Copy-Per-Server` and `One-Copy-Per-Application` Topic Message Distribution Modes set the ClientIDPolicy to Unrestricted. Note that if two WebLogic JMS subscription references on the same physical topic have the same Client ID and durable subscription name, then the references resolve to a single subscription if the Client ID Policy is also the same, but they resolve to two different subscriptions if the Client ID Policies are different.

Copyright © 2007, 2026, Oracle and/or its affiliates.

C

How WebLogic MDBs Leverage WebLogic JMS Extensions

Examine how MDBs utilize certain WebLogic JMS features such as shared subscriptions and unrestricted client IDs.
The MDB deployment scenarios described in Topic Deployment Scenarios take advantage of the following JMS features:

- **Shared subscriptions** -- Shared subscriptions allow multiple subscribers, including subscribers on different MDB servers, to consume from the same subscription on a physical topic or distributed-topic member. Messages available to that subscription are distributed among its subscribers, with each message delivered to one subscriber at a time. Messages may be redelivered following rollback or recovery. For example, if there are two subscribers, S1 and S2, and three messages, M1, M2, and M3, S1 might receive M1 and M2 (but not M3) and, then, S2 would receive M3 (but not M1 and M2).

  Messages can be processed in parallel by the subscribers sharing the subscription. MDBs can create multiple subscribers on the same subscription identifier, whether it is durable or non-durable. For more information about the JMS Subscription Sharing Policy, see Configure Shared Subscriptions in *Administering JMS Resources for Oracle WebLogic Server*.
- **Unrestricted Client IDs** -- Unrestricted Client IDs allow multiple concurrently active connections to use the same Client ID. The JMS `clientID` identifies a JMS connection and is used to identify a durable subscription on that connection. Setting the Client ID Policy to `Unrestricted` allows separate durable subscriptions with the same client ID and subscription name to exist on different physical topics or distributed-topic members. This allows subscriptions with the same name to exist on different members of the same distributed topic, and together, these subscriptions can be treated as a single logical subscription. See Configure an Unrestricted Client ID in *Administering JMS Resources for Oracle WebLogic Server*.

For supported topic MDB configurations using WebLogic Server's built-in JMS integration, the `topicMessagesDistributionMode` activation configuration property determines how the container applies the connection policies shown in Table C-1. These policies control subscription sharing and the use of client IDs.

Table C-1 Connection Policies Used by Topic MDBs

topicMessagesDistributionMode	SubscriptionSharingPolicy	ClientIdPolicy
One-Copy-Per-Server or One-Copy-Per-Application (WebLogic JMS)	Sharable	Unrestricted
Compatibility (WebLogic standalone topics and replicated distributed topics)	Connection factory's policy; Exclusive by default	Connection factory's policy; Restricted by default
Compatibility (foreign topics)	WebLogic policy not applicable	WebLogic policy not applicable

With WebLogic JMS, `Compatibility` mode inherits the connection factory's Subscription Sharing Policy and Client ID Policy. In the One-Copy modes, the container sets the required

Copyright © 2007, 2026, Oracle and/or its affiliates.

policies on the MDB connection instances: `Sharable` for Subscription Sharing Policy and `Unrestricted` for Client ID Policy. The connection-factory configuration is unchanged. In the One-Copy modes, leave the connection factory's Client ID unset so the container can assign the MDB's client ID. See Topic Subscription Identifiers.

Copyright © 2007, 2026, Oracle and/or its affiliates.