Oracle® Fusion Middleware

Administering HTTP Session Management with Oracle Coherence*Web



26 (26.1.0.0.0)
G56899-01
November 2026



Oracle Fusion Middleware Administering HTTP Session Management with Oracle Coherence*Web, 26 (26.1.0.0.0)

G56899-01

Copyright © 2008, 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 Documents i
Conventions ii

1 Introduction to Coherence*Web

Understanding Coherence*Web 1
Supported Web Containers 1
Supported Jakarta Servlet Version 3
Configuration and Deployment Road Map 3
- Choose Your Cluster Node Isolation 3
- Choose Your Locking Mode 3
- Choose How to Scope Sessions and Session Attributes 4
- Choose When to Clean Up Expired HTTP Sessions 4
- Choose the Integration Method 4

2 Using Coherence*Web with WebLogic Server

Overview of Coherence*Web 1
Overview of Managed Coherence Servers 3
Configuring and Deploying Coherence*Web: Main Steps 3
- Summary of Configuring and Deploying Coherence*Web 3
- Installing WebLogic Server and Oracle Coherence 4
- Configure Coherence*Web 4
- Configure the Session Cookies 6
- Start a Cache Server 9
  - Starting a Coherence Cache Server from WebLogic Remote Console 9
  - Starting a Coherence Cache Server from the Command Line 10
- Configure Coherence*Web Storage Mode 11
- Deploying Applications to WebLogic Server 11

Coherence MBean Attributes for Coherence*Web 12

Copyright © 2008, 2026, Oracle and/or its affiliates.

Enabling the Coherence Session Cache in WebLogic Remote Console 12
Using a Custom Session Cache Configuration File 13
Scoping the Session Cookie Path 14
Updating the Session ID 15
Sharing Coherence*Web Sessions with Other Application Servers 16

3 Using Coherence*Web on Other Application Servers

Integrating Coherence*Web Using the WebInstaller 1
General Instructions for Integrating Coherence*Web Session Management Module 2
Deploying and Running Applications In Process 3
Deploying and Running Applications Out-of-Process 4
Migrating to Out-of-Process Topology 4
Deploying and Running Applications Out-of-Process with Coherence*Extend 4
Enabling Sticky Sessions for Apache Tomcat Servers 5
Integrating with IBM WebSphere Liberty 6
Coherence*Web WebInstaller Ant Task 6
Using the Coherence*Web WebInstaller Ant Task 6
Configuring the WebInstaller Ant Task 7
WebInstaller Ant Task Examples 8
Testing HTTP Session Management 8
How the Coherence*Web WebInstaller Instruments a Jakarta EE Application 9
Integrating Coherence*Web with Applications Using Jakarta Security 11
Preventing Cross-Site Scripting Attacks 11
Selecting CoherenceWeb Artifacts and Container APIs 12

4 Tomcat Native Session Integration

Tomcat Configuration 1
Manager Configuration Options 2

5 Coherence*Web Session Management Features

Session Models 1
Overview of Session Models 2
Monolithic Model 3
Traditional Model 5
Split Model 7
Session Model Recommendations 8
Configuring a Session Model 9
Sharing Data in a Clustered Environment 9
Scalability and Performance 10

Copyright © 2008, 2026, Oracle and/or its affiliates.

Session and Session Attribute Scoping 12
Session Scoping 12
Preventing Web Applications from Sharing Session Data 13
Working with Multiple Cache Configurations 14
Keeping Session Cookies Separate 14
Session Attribute Scoping 14
Sharing Session Information Between Multiple Applications 15
Cluster Node Isolation 16
Application Server-Scoped Cluster Nodes 16
EAR-Scoped Cluster Nodes 18
WAR-Scoped Cluster Nodes 19
Session Locking Modes 22
Overview of Session Locking Modes 22
Optimistic Locking 23
Last-Write-Wins Locking 23
Member Locking 23
Application Locking 23
Thread Locking 24
Troubleshooting Locking in HTTP Sessions 24
Enabling Sticky Session Optimizations 25
Deployment Topologies 25
In-Process Topology 25
Out-of-Process Topology 26
Migrating from In-Process to Out-of-Process Topology 27
Out-of-Process with Coherence*Extend Topology 27
Configuring Coherence*Web with Coherence*Extend 28
Overview of Configuring Coherence*Web with Coherence*Extend 28
Configure the Cache for Proxy and Storage JVMs 28
Configure the Cache for Web Tier JVMs 30
Accessing Sessions with Lazy Acquisition 31
Overriding the Distribution of HTTP Sessions and Attributes 31
Overview of Overriding HTTP Session Distribution 31
Implementing a Session Distribution Controller 32
Registering a Session Distribution Controller Implementation 33
Detecting Changed Attribute Values 33
Saving Non-Serializable Attributes Locally 33
Securing Coherence*Web Deployments 33
Customizing the Name of the Session Cache Configuration File 34
Configuring Logging for Coherence*Web 35
Getting Concurrent Access to the Same Session Instance 35
Federated Session Caches 36

Copyright © 2008, 2026, Oracle and/or its affiliates.

Preparing Session Attributes for Hardened Mode 37

6 Monitoring Applications

Managing and Monitoring Applications with JMX 1
Managing and Monitoring Applications on WebLogic Server 4
Running Performance Reports 5
Web Session Cache Storage Report 5
Web Session Cache Overflow Report 7
Web Report 9
WebLogic Web Report 10
Web Service Report 10

7 Cleaning Up Expired HTTP Sessions

Understanding the Session Reaper 1
Tuning the Session Reaper 4
Getting Session Reaper Performance Statistics 4
Understanding Session Invalidation Exceptions for the Session Reaper 5
Verifying Session Expiry Failover and Cleanup 5

8 Working with JSF and MyFaces Applications

Configuring for all JSF and MyFaces Web Applications: 1
Configuring for Instrumented Applications that use MyFaces 2
Configuring for Instrumented Applications that use Mojarra 2

A Coherence*Web Context Parameters

B Capacity Planning

Sizing Modern Session Storage B-2

C Session Cache Configuration File

Copyright © 2008, 2026, Oracle and/or its affiliates.

Preface

*Administering HTTP Session Management with Oracle Coherence*Web* describes how to deploy Oracle Coherence*Web (Coherence*Web), an HTTP session management module to WebLogic Server and other application servers. It also describes the different session management features that you can configure.

This preface includes the following sections:

- Audience
- Documentation Accessibility
- Diversity and Inclusion
- Related Documents
- Conventions

Audience

This guide is intended for application developers who want to be able to manage session state in clustered environments.

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 Documents

For more information, see the following in the Oracle Coherence documentation set:

Copyright © 2008, 2026, Oracle and/or its affiliates.

- *Installing Oracle Coherence*
- *Release Notes for Oracle Coherence*
- *Managing Oracle Coherence*
- *Developing Applications with Oracle Coherence*
- *Developing Oracle Coherence Applications for Oracle WebLogic Server*
- *Securing Oracle Coherence*
- *Integrating Oracle Coherence*
- *Administering HTTP Session Management with Oracle Coherence*Web*
- *Developing Remote Clients for Oracle Coherence*
- *Java API Reference for Oracle Coherence*
- *C++ API Reference for Oracle Coherence*
- *.NET API Reference for Oracle Coherence*
- *REST API for Managing Oracle Coherence*

Conventions

The following text conventions are used in this document:

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 which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Copyright © 2008, 2026, Oracle and/or its affiliates.

1

Introduction to Coherence*Web

Learn the advantages of using Coherence*Web for managing session state in clustered environments. Coherence*Web can be easily configured and used with many containers. More detailed information on configuration, deployment, and features are provided in following chapters.

This chapter includes the following sections:

- Understanding Coherence*Web
- Supported Web Containers
- Supported Jakarta Servlet Version
- Configuration and Deployment Road Map

Understanding Coherence*Web

Coherence*Web is an HTTP session management module dedicated to managing session state in clustered environments.
Built on top of Oracle Coherence, Coherence*Web:

- Brings Coherence data grid's data scalability, availability, reliability, and performance to in-memory session management and storage.
- Can configure fine-grained session and session attribute scoping by way of pluggable policies. See Session and Session Attribute Scoping.
- Can be deployed to many mainstream application servers such as Oracle WebLogic Server, IBM WebSphere, and Tomcat. See Supported Web Containers.
- Allows storage of session data outside of the Jakarta EE application server, freeing application server heap space and enabling server restarts without session data loss. See Deployment Topologies.
- Enables session sharing and management across different Web applications, domains and heterogeneous application servers. See Session and Session Attribute Scoping.
- Can be used in advanced session models (that is, Monolithic, Traditional, and Split Session) that define how the session state is serialized or deserialized in the cluster. See Session Models.

Supported Web Containers

Coherence*Web is supported on many platforms like WebLogic Server, WebSphere Liberty, Tomcat, Wildfly, and Jetty. In the recent releases of WebLogic Server, Coherence*Web is integrated with it and does not require installation or special integration steps.
See Using Coherence*Web with WebLogic Server.

For third-party application servers, Coherence*Web provides a generic utility, the *WebInstaller*, that transparently instruments your Web applications. Using Coherence*Web on Other Application Servers , describes how to use the WebInstaller to integrate Coherence*Web with these servers.

Copyright © 2008, 2026, Oracle and/or its affiliates.

> **Note**
>
> When migrating from a javax.servlet-based application, review namespace, deployment descriptors, libraries and container integration together. Rebuild and test against the selected jakarta.servlet API; changing imports alone does not establish compatibility. Java SE javax packages are unchanged.

Table 1-1 summarizes the Web containers supported by the Coherence*Web session management module. It also provides links to the information required to integrate Coherence*Web. Notice that all of the Web containers (except Oracle WebLogic Server) share the same general instructions.

> **Note**
>
> The value in the Server Type Alias column is used only by the Coherence*Web WebInstaller. The value is passed to the WebInstaller through the `-server` command-line option.

Table 1-1 Web Containers which can use Coherence*Web

Application Server	Server Type Alias	See this Integration Section
Oracle WebLogic	N/A	Coherence*Web is integrated with WebLogic Server. No integration steps are required. See Using Coherence*Web with WebLogic Server. **Note** WebLogic Server and Coherence must be on the same versions when using Coherence*Web.
Application Server: Open Liberty 23.0.0.5 Server Type Alias: Open Liberty 23.0.0.5	`Liberty/Servlet5.0` `Liberty/Servlet6.0`	General Instructions for Integrating Coherence*Web Session Management Module
Apache Tomcat 10.0.*n*	`Tomcat/Servlet5.0`	General Instructions for Integrating Coherence*Web Session Management Module and Enabling Sticky Sessions for Apache Tomcat Servers
Apache Tomcat 10.1.*n*	`Tomcat/Servlet6.0`	General Instructions for Integrating Coherence*Web Session Management Module and Enabling Sticky Sessions for Apache Tomcat Servers
Apache Tomcat 11.0.*x*	`Tomcat/Servlet6.1`	General Instructions for Integrating Coherence*Web Session Management Module and Enabling Sticky Sessions for Apache Tomcat Servers
Jetty 12.*n*	`Jetty/Servlet5.0` `Jetty/Servlet6.0`	General Instructions for Integrating Coherence*Web Session Management Module
Wildfly 33.0.*x*.Final/ Undertow	`Wildfly/Servlet6.0`	General Instructions for Integrating Coherence*Web Session Management Module

Copyright © 2008, 2026, Oracle and/or its affiliates.

Application requirements

Before deployment, complete Selecting CoherenceWeb Artifacts and Container APIs.
Container API level, native integration and distribution certification are separate checks. Keep the container-provided Servlet API and lifecycle ownership intact.

Supported Jakarta Servlet Version

Coherence*Web supports only the APIs up to Jakarta Servlet 6.1.
Any API in later versions of Jakarta Servlet supported by Coherence*Web will be explicitly documented.

Before deployment, complete Selecting CoherenceWeb Artifacts and Container APIs.
Container API level, native integration and distribution certification are separate checks. Keep the container-provided Servlet API and lifecycle ownership intact.

Configuration and Deployment Road Map

Coherence*Web includes configuration options that might need to change depending on your environment and session requirements.
This section includes the following topics:

- Choose Your Cluster Node Isolation
- Choose Your Locking Mode
- Choose How to Scope Sessions and Session Attributes
- Choose When to Clean Up Expired HTTP Sessions
- Choose the Integration Method

Choose Your Cluster Node Isolation

Cluster node isolation refers to the scope of the Coherence nodes that are created within each application server JVM. Several different isolation modes are supported.

For example: you might be deploying multiple applications to the container that require the use of the same cluster (or one Coherence node); you might have multiple Web applications packaged in a single EAR file that use a single cluster; or you might have Web applications that must keep their session data separate and must be deployed to their own individual Coherence cluster. These choices and the deployment descriptors and elements that must be configured are described in Cluster Node Isolation.

Choose Your Locking Mode

Locking mode refers to the behavior of HTTP sessions when they are accessed concurrently by multiple Web container threads. Coherence*Web offers several different session locking options. For example, you can allow multiple nodes in a cluster to access an HTTP session simultaneously, or allow only one thread at a time to access an HTTP session. You can also allow multiple threads to access the same Web application instance while prohibiting concurrent access by threads in different Web application instances. These choices, and the deployment descriptors and elements that must be configured, are described in Session Locking Modes.

Copyright © 2008, 2026, Oracle and/or its affiliates.

Choose How to Scope Sessions and Session Attributes

Session and session attribute scoping refers to the fine-grained control over how both session data and session attributes are scoped (or *shared*) across application boundaries. Coherence*Web supports sharing sessions across Web applications and restricts which session attributes are shared across the application boundaries. These choices, and the deployment descriptors and elements that must be configured, are described in Session and Session Attribute Scoping.

Choose When to Clean Up Expired HTTP Sessions

Coherence*Web provides a session reaper, which invalidates sessions that have expired. Cleaning Up Expired HTTP Sessions , describes the session reaper.

Choose the Integration Method

The integration procedure that you follow depends on your application server. Supported Web Containers provides a list of the application servers and the corresponding instructions for integrating Coherence*Web.

If you are using a recent release of WebLogic Server, Coherence and Coherence*Web are installed with the WebLogic Server product. No separate Coherence*Web integration is necessary. See Using Coherence*Web with WebLogic Server.

For other application servers, use the generic Jakarta EE WebInstaller described in Using Coherence*Web on Other Application Servers .

If using Apache Tomcat, developers may opt to forgo the installer and choose to use Coherence*Web's native session integration. Applications may be deployed as is. For more information, see Tomcat Native Session Integration.

Application requirements

Before deployment, complete Selecting CoherenceWeb Artifacts and Container APIs. Container API level, native integration and distribution certification are separate checks. Keep the container-provided Servlet API and lifecycle ownership intact.

Copyright © 2008, 2026, Oracle and/or its affiliates.

2

Using Coherence*Web with WebLogic Server

The Coherence*Web module for WebLogic Server simplifies session state persistence deployment, configuration, and management. This chapter provides an overview of Managed Coherence Servers and the Grid Archive (GAR) format for packaging Coherence applications. A detailed discussion of Managed Coherence Servers and the GAR format is beyond the scope of this document. See *Oracle Fusion Middleware Developing Oracle Coherence Applications for Oracle WebLogic Server*.
This chapter includes the following sections:

Before deployment, complete Selecting CoherenceWeb Artifacts and Container APIs. Container API level, native integration and distribution certification are separate checks. Keep the container-provided Servlet API and lifecycle ownership intact.

- Overview of Coherence*Web
  Coherence*Web provides session state persistence and management. It is a session management module that uses Coherence caches for storing and managing session data.
- Overview of Managed Coherence Servers
- Configuring and Deploying Coherence*Web: Main Steps
  You can configure and deploy Coherence*Web with applications running on WebLogic Server.
- Coherence MBean Attributes for Coherence*Web
- Enabling the Coherence Session Cache in WebLogic Remote Console
- Using a Custom Session Cache Configuration File
- Scoping the Session Cookie Path
  WebLogic Server and Coherence*Web handle session scoping and the session lifecycle in different ways. This can impact your decision to implement a single sign-on (SSO) strategy for your applications.
- Updating the Session ID
  When a user successfully authenticates a protected resource, the session ID is changed for security purposes.
- Sharing Coherence*Web Sessions with Other Application Servers

Overview of Coherence*Web

Coherence*Web provides session state persistence and management. It is a session management module that uses Coherence caches for storing and managing session data.

Coherence*Web is an alternative to the WebLogic Server in-memory HTTP state replication services. Consider using Coherence*Web if you are encountering any of these situations:

- Your application works with large HTTP session state objects
- You run into memory constraints, due to storing HTTP session object data
- You want to offload HTTP session storage to an existing Coherence cluster
- You want to share session state across enterprise applications and Web modules

Copyright © 2008, 2026, Oracle and/or its affiliates.

The classes that define the Coherence*Web are contained in the `coherence-web.jar` file. To use the functionality provided by Coherence*Web, the `coherence.jar` classes must also be available to the Web application. Both of these libraries are on the WebLogic Server system classpath and are automatically loaded at runtime. The `coherence-web.jar` loads application classes with the appropriate classloader in WebLogic Server.

> **Note**
>
> WebLogic Server and Coherence must be on the same versions when using Coherence*Web.

Coherence cache configurations and services used by Coherence*Web are defined in the `default-session-cache-config.xml` file, which can be found in the `coherence-web.jar` file. The default cache and services configuration defined in the `default-session-cache-config.xml` file should satisfy most Web applications.

You can create your own custom session cache configuration by packaging a file named `session-cache-config.xml` in your Web application. See Using a Custom Session Cache Configuration File.

When Coherence*Web is started on WebLogic Server, it first looks for a file named `session-cache-config.xml`. For example, the file can be placed in a WAR file's `WEB-INF/classes` directory, or packaged in a JAR file and placed in an EAR file's `APP-INF/lib` directory. If no custom session cache configuration XML resource is found, then it will use the `default-session-cache-config.xml` file packaged in `coherence-web.jar`.

In Coherence*Web, the following default cache configurations are defined:

- Coherence*Web for WebLogic Server is configured with local-storage disabled. The server will serve requests and will not be used to host data. This means a Coherence cache server must be running in its own JVM, separate from the JVM running WebLogic Server.
- The timeout for requests to the cache server to respond is 30 seconds. If a request to the cache server has not responded in 30 seconds, a `com.tangosol.net.RequestTimeoutException` exception is thrown.

All Coherence*Web-enabled applications running on WebLogic Server have application server-scope. In this configuration, all deployed applications become part of one Coherence node. See Cluster Node Isolation.

Coherence*Web provides several session locking modes to control concurrent access of sessions. Coherence*Web employs Last Write Wins locking by default. See Session Locking Modes.

By itself, Coherence*Web does not require a load balancer to run in front of the WebLogic Server tier. However, a load balancer will improve performance. It is required if the same session will be used concurrently and locking is not enabled. The default load balancer enforces HTTP session JVM affinity, however, other load balancing alternatives are available. WebLogic Server ships with several different proxy plug-ins which enforce JVM session stickiness. For information on configuring the WebLogic Server proxy plug-in, see Load Balancing with a Proxy Plug-in in *Administering Clusters for Oracle WebLogic Server*.

Copyright © 2008, 2026, Oracle and/or its affiliates.

Overview of Managed Coherence Servers

Oracle WebLogic Server and Coherence have defined Managed Coherence Servers which provide Coherence applications with the same benefits as other Jakarta EE applications that are hosted on WebLogic Server.
The following are the examples of Coherence applications functioning similar to Jakarta EE applications:

- Coherence applications can be deployed in a manner similar to other Jakarta EE applications.
- Coherence applications in the grid can be managed using the standard management tools included with WebLogic Server.
- Coherence clusters can be configured by using WebLogic configuration.
- Coherence Grid Archives can be integrated into Enterprise Archives (EAR files).
- Coherence applications can integrate with existing Coherence-based functionality.

> **Note**
>
> Using multiple Coherence clusters in a single WebLogic Server domain is not recommended.

For more information on Managed Coherence Servers, see Creating Coherence Applications for WebLogic Server in *Developing Oracle Coherence Applications for Oracle WebLogic Server*.

Configuring and Deploying Coherence*Web: Main Steps

You can configure and deploy Coherence*Web with applications running on WebLogic Server.

This section includes the following topics:

- Summary of Configuring and Deploying Coherence*Web
- Installing WebLogic Server and Oracle Coherence
- Configure Coherence*Web
- Configure the Session Cookies
- Start a Cache Server
- Configure Coherence*Web Storage Mode
- Deploying Applications to WebLogic Server

Summary of Configuring and Deploying Coherence*Web

The following steps summarize how to prepare your deployments to use Coherence*Web with applications running on WebLogic Server:

1. Install WebLogic Server and Oracle Coherence. See Installing WebLogic Server and Oracle Coherence.

Copyright © 2008, 2026, Oracle and/or its affiliates.

2. (Optional) Modify the `web.xml` file in the deployment if your application requires advanced configuration for Coherence*Web. Configure Coherence*Web describes the parameters that can be configured for Web applications. The entire set of Coherence*Web parameters are described in Coherence*Web Context Parameters.
3. (Optional) Configure the WebLogic-generated HTTP session cookie parameters in the `weblogic.xml` or `weblogic-application.xml` file. See Configure the Session Cookies.
4. (Optional for testing; strongly suggested for production) Start a Cache Server Tier in a separate JVM from the one running WebLogic Server. See Start a Cache Server.
5. Set the Coherence*Web storage mode. See Configure Coherence*Web Storage Mode.
6. Deploy the application to WebLogic Server. See Deploying Applications to WebLogic Server.

Installing WebLogic Server and Oracle Coherence

WebLogic Server is installed by executing its installer. The installer provides the full installation and allows you to individually select the components to install (bits, examples, Javadoc, and so on). The installer supports both a graphical mode using the Oracle Universal Installer (OUI) and a silent mode. Installing Coherence is an option in the WebLogic Server installer.

WebLogic Server is always installed to the *`ORACLE_HOME`*`/wlserver` directory; Coherence is always installed to the `ORACLE_HOME/coherence` directory.

See *Installing and Configuring Oracle WebLogic Server and Coherence*.

Configure Coherence*Web

Coherence*Web provides a default configuration that should satisfy most Web applications. Table 2-1 describes the context parameters configured by Coherence*Web. Table 2-2 describes the compatibility mode context parameter. For complete descriptions of all Coherence*Web parameters, see Coherence*Web Context Parameters.

You can also configure the context parameters on the command line as system properties. The system properties have the same name as the context parameters, but the dash (`-`) is replaced with a period (`.`). For example, to declare a value for the context parameter `coherence-enable-sessioncontext` on the command line, enter it like this:

```
-Dcoherence.enable.sessioncontext=true
```

If both a system property and the equivalent context parameter are configured, the value from the system property is used.

Copyright © 2008, 2026, Oracle and/or its affiliates.

Table 2-1 Context Parameters Configured by Coherence*Web

Parameter Name	Description
`coherence-application-name`	Coherence*Web uses the value of this parameter to determine the name of the application that uses the `ApplicationScopeController` interface to scope attributes. The value for this parameter should be provided in the following format: *application name* + `!` + *Web module name* The *application name* is the name of the application that uses the `ApplicationScopeController` interface and *Web module name* is the name of the Web module in which it appears. For example, if you have an EAR file named `test.ear` and a Web-module named `app1` defined in the EAR file, then the default value for the `coherence-application-name` parameter would be `test!app1`. If this parameter is not configured, then Coherence*Web uses the name of the class loader instead. Also, if the parameter is not configured and the `ApplicationScopeController` interface *is* configured, then a warning is logged saying that the application name was not configured. See Session Attribute Scoping for more information.
`coherence-reaperdaemon-assume-locality`	This setting allows the session reaper to assume that the sessions that are stored on this node (for example, by a distributed cache service) are the only sessions that this node must check for expiration. The default is `false`.
`coherence-scopecontroller-class`	This value specifies the class name of the optional `com.tangosol.coherence.servlet.HttpSessionCollection$AttributeScopeController` interface implementation. Valid values include: • `com.tangosol.coherence.servlet.AbstractHttpSessionCollection$ApplicationScopeController` (default) • `com.tangosol.coherence.servlet.AbstractHttpSessionCollection$GlobalScopeController` The default set by Coherence*Web is `com.tangosol.coherence.servlet.AbstractHttpSessionCollection$ApplicationScopeController`.

Table 2-2 describes the `coherence-session-weblogic-compatibility-mode` context parameter which is specifically provided by Coherence*Web.

Table 2-2 Context Parameter Provided by the Coherence*Web

Parameter Name	Description
`coherence-session-weblogic-compatibility-mode`	This parameter is provided by Coherence*Web. If its value is set to `true`, it determines that a single session ID (with the cookie path set to "`/`") will map to a unique Coherence*Web session instance in each Web application. If it is `false`, then the standard behavior will apply: a single session ID will map to a single session instance using Coherence*Web in WebLogic Server. All other session persistence mechanisms in WebLogic use a single session ID in each Web application to refer to different session instances. This parameter defaults to `true` unless the global scope controller is specified. If this controller is specified, then the parameter defaults to `false`.

Table 2-3 describes the `coherence-factory-class` context parameter. The default value, which is set by Coherence*Web, should not be changed.

Copyright © 2008, 2026, Oracle and/or its affiliates.

Table 2-3 Context Parameter Value that Should Not be Changed

Parameter Name	Description
`coherence-factory-class`	The fully qualified name of the class that implements the `SessionHelper.Factory` factory class. Coherence*Web sets the default value to `weblogic.servlet.internal.session.WebLogicSPIFactory`. This value should not be changed.

Configure the Session Cookies

If you are using Coherence*Web, then WebLogic Server generates and parses the session cookie. In this case, any native Coherence*Web session cookie configuration parameters will be ignored. To configure the session cookies, use the WebLogic-generated HTTP session cookie parameters in the `weblogic.xml` or `weblogic-application.xml` files. Table 2-4 describes these parameters.

In this table, **Updatable?** indicates whether the value of the parameter can be changed while the server is running. **Not applicable** indicates that there is no corresponding Coherence session cookie parameter.

Table 2-4 WebLogic-Generated HTTP Session Cookie Parameters

This Session Cookie Parameter...	Replaces this Coherence*Web Cookie Parameter	Description
`cookie-comment`	**Not applicable**	Specifies the comment that identifies the session tracking cookie in the cookie file. The default is `null`. **Updatable?** Yes
`cookie-domain`	`coherence-session-cookie-domain`	Specifies the domain for which the cookie is valid. For example, setting `cookie-domain` to `.example.com` returns cookies to any server in the `*.example.com` domain. The domain name must have at least two components. Setting a name to `*.com` or `*.net` is not valid. If not set, this attribute defaults to the server that issued the cookie. For more information, see `Cookie.setDomain()` in the Servlet specification. The default is `null`. **Updatable?** Yes
`cookie-max-age-secs`	`coherence-session-max-age`	Sets the life span of the session cookie, in seconds, after which it expires on the client. See Using Sessions and Session Persistence in *Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server*. The default value is `-1` (unlimited). **Updatable?** Yes
`cookie-name`	`coherence-session-cookie-name`	Defines the session-tracking cookie name. Defaults to `JSESSIONID` if not set. You can set this to a more specific name for your application. The default is `JSESSIONID`. **Updatable?** Yes

Copyright © 2008, 2026, Oracle and/or its affiliates.

Table 2-4 (Cont.) WebLogic-Generated HTTP Session Cookie Parameters

This Session Cookie Parameter...	Replaces this Coherence*Web Cookie Parameter	Description
`cookie-path`	`coherence-session-cookie-path`	Defines the session-tracking cookie path. If not set, this attribute defaults to a slash ("`/`") where the browser sends cookies to all URLs served by WebLogic Server. You can set the path to a narrower mapping, to limit the request URLs to which the browser sends cookies. The default is `null`. **Updatable?** Yes
`cookie-secure`	`coherence-session-cookie-secure`	Tells the browser that the cookie can be returned only over an HTTPS connection. This ensures that the cookie ID is secure and should be used only on Web sites that use HTTPS. Session cookies sent over HTTP will not work if this feature is enabled. Disable the `url-rewriting-enabled` element if you intend to use this feature. WebLogic Server generates the session cookie. The default is `false`. **Updatable?** Yes
`cookies-enabled`	`coherence-session-cookies-enabled`	Enables use of session cookies by default and is recommended, but you can disable them by setting this property to `false`. You might turn this option off for testing purposes. The default is `true`. **Updatable?** Yes
`debug-enabled`	**Not applicable**	Enables the debugging feature for HTTP sessions. Support it by enabling `HttpSessionDebug` logging and the WebLogic Server trace logger. The default value is `false`. **Updatable?** Yes
`encode-session-id-in-query-params`	**Not applicable**	Is set to `true` if the latest servlet specification requires containers to encode the session ID in path parameters. Certain Web servers do not work well with path parameters. In such cases, the `encode-session-id-in-query-params` element should be set to `true`. WebLogic Server generates the HTTP response. The default value is `false`. **Updatable?** Yes
`http-proxy-caching-of-cookies`	**Not applicable**	When set to `false`, WebLogic Server adds the following header and response to indicate that the proxy caches are not caching the cookies: "`Cache-control: no-cache=set-cookie`" WebLogic Server generates the HTTP response. The default value is `true.` **Updatable?** Yes

Copyright © 2008, 2026, Oracle and/or its affiliates.

Table 2-4 (Cont.) WebLogic-Generated HTTP Session Cookie Parameters

This Session Cookie Parameter...	Replaces this Coherence*Web Cookie Parameter	Description
`id-length`	`coherence-session-id-length`	Sets the size of the session ID. The minimum value is 8 bytes and the maximum value is `Integer.MAX_VALUE`. If you are writing a Wireless Application Protocol (WAP) application, you must use URL rewriting because the WAP protocol does not support cookies. Also, some WAP devices have a 128-character limit on URL length (including attributes), which limits the amount of data that can be transmitted using URL rewriting. To allow more space for attributes, use this attribute to limit the size of the session ID that is randomly generated by WebLogic Server. You can also limit the length to a fixed 52 characters, and disallow special characters, by setting the `WAPEnabled` attribute. See URL Rewriting and Wireless Access Protocol in *Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server* The default is 52. **Updatable?** No
`invalidation-interval-secs`	`coherence-reaperdaemon-cycle-seconds`	Sets the time, in seconds, that Coherence*Web waits between checks for timed-out and invalid sessions, and deleting the old sessions and freeing up memory. Use this element to tune WebLogic Server for best performance on high traffic sites. The default is `60`. **Updatable?** No
`timeout-secs`	**Not applicable**	Sets the time, in seconds, that Coherence*Web waits before timing out a session. On busy sites, you can tune your application by adjusting the timeout of sessions. While you want to give a browser client every opportunity to finish a session, you do not want to tie up the server needlessly if the user has left the site or otherwise abandoned the session. This element can be overridden by the `session-timeout` element (defined in minutes) in `web.xml`. The default is `3600` seconds. **Updatable?** No
`tracking-enabled`	**Not applicable**	Enables session tracking between HTTP requests. WebLogic Server generates the HTTP response. The default is `true`. **Updatable?** No
`url-rewriting-enabled`	`coherence-session-urlencode-enabled`	Enables URL rewriting, which encodes the session ID into the URL and provides session tracking if cookies are disabled in the browser and the `encodeURL` or `encodeRedirectedURL` methods are used when writing out URLs. For more information, see: `http://www.jguru.com/faq/view.jsp?EID=1045` WebLogic Server generates the HTTP response. The default is `true`. **Updatable?** Yes

Copyright © 2008, 2026, Oracle and/or its affiliates.

> **Note**
>
> The default for WebLogic HTTP Session parameter `invalidation-interval-secs` of 60 seconds overrides the Coherence*Web Session Cookie parameter `coherence-reaperdaemon-cycle-seconds` default of 5 minutes.

Start a Cache Server

A Coherence cache server is responsible for storing and managing all cached data. Coherence is integrated within WebLogic Server as a container subsystem. The use of a container aligns the lifecycle of a Coherence cluster member with the lifecycle of a managed server: starting or stopping a managed server JVM starts and stops a Coherence cluster member. Managed servers that are cluster members are referred to as managed Coherence servers.

Coherence clusters are different than WebLogic Server clusters. They use different clustering protocols and are configured separately. Multiple WebLogic Server clusters can be associated with a Coherence cluster and a WebLogic Server domain should only contain a single Coherence cluster. Managed Coherence servers can be explicitly associated with a Coherence cluster or they can be associated with a WebLogic Server cluster that is associated with a Coherence cluster. WebLogic Server managed servers that are members of a Coherence cluster and are storage-enabled, act as cache servers. See Configuring and Managing Coherence Clusters in *Administering Clusters for Oracle WebLogic Server*.

You can start a Coherence cache server or cluster either from the WebLogic Remote Console or from the command line, as described in the following sections.

- Starting a Coherence Cache Server from WebLogic Remote Console
- Starting a Coherence Cache Server from the Command Line

Starting a Coherence Cache Server from WebLogic Remote Console

Using the WebLogic Remote Console, you can enable storage for each WebLogic Server that is a member of a Coherence cluster. The Coherence session caches have a separate flag for enabling storage. See Enabling the Coherence Session Cache in WebLogic Remote Console .

> **Note**
>
> If your managed server is a member of a Coherence cluster and is using Coherence*Web, then you can enable session storage by adding the `-Dcoherence.session.localstorage=true` system property to the startup command.

Coherence session caches automatically start with the WebLogic Server cluster.

The following steps summarize how to start a Coherence cluster in the WebLogic Remote Console.

1. Configure the Coherence Cluster.

   See Configuring and Managing Coherence Clusters in *Administering Clusters for Oracle WebLogic Server*.

2. Configure WebLogic Servers and clusters that will be associated with the Coherence cluster.

Copyright © 2008, 2026, Oracle and/or its affiliates.

See Configuring and Managing Coherence Clusters in *Administering Clusters for Oracle WebLogic Server*.

3. Enable Coherence*Web for the selected WebLogic Servers or clusters.

   See Enabling the Coherence Session Cache in WebLogic Remote Console .

Starting a Coherence Cache Server from the Command Line

Instead of using the WebLogic Remote Console, there may be situations when you might need to start a Coherence cache server or cluster from the command line. You can start the Coherence cache server from the command line either in standalone mode, or as part of a WebLogic Server instance.

This section includes the following topics:

- To Start a Standalone Coherence Cache Server
- To Start a Storage-Enabled or -Disabled WebLogic Server Instance

To Start a Standalone Coherence Cache Server

Follow these steps to start a standalone Coherence cache server:

1. Create a script for starting a Coherence cache server. The following is a simple example of a script that creates and starts a storage-enabled cache server. This example assumes that you are using a Sun JVM. See JVM Tuning in *Administering Oracle Coherence* for more information.

   ```
   java -cp "$COHERENCE_HOME/lib/
   coherence.jar:$COHERENCE_WEB_JAR:$SESSION_MODEL_CLASSPATH" \
     -Dcoherence.cluster="$SESSION_CLUSTER" \
     -Dcoherence.cacheconfig="$SESSION_CACHE_CONFIG" \
     -Dcoherence.session.localstorage=true \
     com.tangosol.net.Coherence
   ```

   You must include `coherence-web.jar` and `coherence.jar` on the classpath. The variable *`session_cache_configuration_file`* represents the absolute path to the cache configuration file on your file system. For Coherence*Web, the default session cache configuration file is named `default-session-cache-config.xml`. Note that the cache configuration defined for the cache server must match the cache configuration defined for the application servers which run on the same Coherence cluster.

   If your application uses additional Coherence caches, then you must merge the cache configuration information with a customized session cache configuration file. This customized session cache configuration file, typically named `session-cache-config.xml`, should contain the contents of `default-session-cache-config.xml` file and the additional caches used by your application.

   The cache and session configuration must be consistent across WebLogic Server and Coherence cache servers.

   For more information on merging these files, see Merging Coherence Cache and Session Information in *Integrating Oracle Coherence*.

   The variable *`Coherence_cluster_name`* represents the name of the Coherence cluster. A cluster name check has been added to 10.3.6 and later versions of WebLogic Server. The `coherence.cluster` property must be added to the cache server because you are declaring the cluster name in the WebLogic Server application. If the Coherence servers are started in standalone mode, they must pass this property, otherwise the cluster will not form between the WLS servers and the standalone cache server.

Copyright © 2008, 2026, Oracle and/or its affiliates.

2. Start one or more Coherence cache servers using the script described in the previous step.

Set COHERENCE_WEB_JAR to the matching selected session module, SESSION_MODEL_CLASSPATH to application model/resources and generated metadata, SESSION_CLUSTER to the deployment cluster and SESSION_CACHE_CONFIG to the complete cache configuration. Use the selected platform classpath separator (semicolon on Windows). Supply the production discovery/transport configuration before starting; use a unique isolated cluster for testing. Size heap/native memory from measurement rather than retaining the older 512 MB example as a recommendation. Enable management only with its intended protection.

To Start a Storage-Enabled or -Disabled WebLogic Server Instance

By default, a Coherence*Web-enabled WebLogic Server instance starts in storage-disabled mode. To start the WebLogic Server instance in storage-enabled mode, follow these steps:

1. Create a script for starting a Coherence cache server. This can be similar to the script described in the previous section.
2. Include the command-line property to enable local storage, `-Dcoherence.session.localstorage=true`, in the server startup command. The WebLogic Server instance will start with Coherence*Web-enabled and local storage enabled.

   To start a Coherence*Web-enabled WebLogic Server instance, omit this system property. Local storage will be disabled by default.

See weblogic.Server Command-Line Reference in *Command Reference for Oracle WebLogic Server*.

Configure Coherence*Web Storage Mode

You can enable Coherence*Web session storage by specifying `coherence-web` as the value of the `persistent-store-type` attribute in the `weblogic.xml` session configuration. This configuration provides application server-level cluster node scoping for web applications deployed on WebLogic Server. No shared libraries need to be deployed or depended upon.

Coherence*Web is initialized only when a web application that requires session persistence is started in the WebLogic Server instance.

Example 2-1 illustrates a sample `weblogic.xml` file where `coherence-web` is the value of the `persistent-store-type` attribute.

Example 2-1 Coherence Web Storage Mode in weblogic.xml

```
<weblogic-web-app>
  ...
<session-descriptor>
      <persistent-store-type>coherence-web</persistent-store-type>
</session-descriptor>
  ...
</weblogic-web-app>
```

Deploying Applications to WebLogic Server

If you are using the default session cache configuration file with your web application, then you can package and deploy it like any other Jakarta EE application. However, if you are using a custom session cache configuration file, then you must package and deploy the application in a GAR file.

Copyright © 2008, 2026, Oracle and/or its affiliates.

GAR files deploy like any other Jakarta EE application, except that you create a Coherence tier and nodes belonging to the tier. You can configure and deploy a standalone GAR or an embedded GAR.

See Deploying Coherence Applications to WebLogic Server in *Administering Oracle Coherence* and Creating Coherence Applications for WebLogic Server in *Developing Oracle Coherence Applications for Oracle WebLogic Server*.

Coherence MBean Attributes for Coherence*Web

WebLogic Server defines a cluster MBean (`weblogic.management.configuration.ClusterMBean`) which represents a cluster in the domain.
The cluster MBean defines a number of attributes, operations, and MBeans related to the management of the cluster. Among the MBeans defined by the cluster MBean are the `CoherenceMemberConfigMBean` and the `CoherenceTierMBean` MBeans.

The `CoherenceMemberConfigMBean` and the `CoherenceTierMBean` MBeans each define an `isCoherenceWebLocalStorageEnabled` attribute that indicates whether a cluster or member is acting as a storage tier for Coherence*Web. This attribute is defined in Table 2-5.

Table 2-5 Coherence MBean Attribute for Coherence*Web

Attribute	Description
isCoherenceWebLocalStorageEnabled	If this attribute is set to `true` in `CoherenceTierMBean`, it indicates that a cluster is acting as a storage tier for Coherence*Web. Coherence*Web cache services will start with storage enabled when the server starts. When deploying a Coherence*Web-enabled application, there must be a running WebLogic cluster in the domain which has this attribute enabled. If this attribute is set to `true` in `CoherenceMemberConfigMBean`, it indicates that this node is acting as a storage node for Coherence*Web. Coherence*Web cache services will start with storage enabled when the server starts. When deploying a Coherence*Web-enabled application, there must be a running WebLogic cluster in the domain which has this attribute enabled. **Default:** `false`

Enabling the Coherence Session Cache in WebLogic Remote Console

The **Coherence Web Local Storage Enabled** and **Coherence Web Federated Storage Enabled** options in the WebLogic Remote Console can be used to indicate whether the cluster is acting as a storage tier for Coherence*Web.
When selecting the federated storage option, the default federation topology which is configured is used. See Configuring Cache Federation in *Administering Clusters for Oracle WebLogic Server* and Federated Session Caches.

1. In the **Edit Tree**, go to **Environment**, then **Clusters**.
2. Select a cluster from the **Clusters** table.

Copyright © 2008, 2026, Oracle and/or its affiliates.

3. Click the **Coherence** tab.
4. Enable or disable the **Coherence Web Local Storage Enabled** and **Coherence Web Federated Storage Enabled** options as desired.
5. Click **Save**.

Using a Custom Session Cache Configuration File

Custom session cache configuration files must be packaged in a GAR file for deployment. The `coherence-web.jar` file contains a `default-session-cache-config.xml` cache configuration file which should be sufficient for most applications. However, if you are working with technologies such as Coherence*Extend or cluster Federation, or if you have WebLogic Server nodes that are to act as storage-enabled cache servers with a custom session cache configuration, then you must provide a custom session cache configuration file. Custom session cache configuration files must be packaged in a GAR file for deployment.

To use a custom session cache configuration file on WebLogic Server and package it in a GAR file, follow these steps for web applications and for the WebLogic Server nodes acting as cache servers:

For web applications using Coherence*Web:

1. If you are using a custom session cache configuration file (which should be named `session-cache-config.xml`), then package it in your web application:
   - For a WAR file, place the session cache configuration file in the `WEB-INF/classes` folder
   - For an EAR file, package the session cache configuration file in a JAR file and place it in the shared library (the `APP-INF/lib` folder) in an EAR file

   Note that you can customize the session cache configuration file name, but then you must provide the new file name as the value of the `coherence-cache-configuration-path` context parameter in the `web.xml` file.
2. If you do not want the WebLogic Server cluster members running the Coherence*Web application to act as a cache server, then ensure that the **Coherence Web Local Storage Enabled** option is disabled. In the WebLogic Remote Console, go to the Edit Tree, then Environment, then Clusters, then *myCluster*, then the Coherence tab. This will cause the custom session cache configuration file to be read.

For WebLogic Server nodes acting as cache servers:

1. If you are using a custom session cache configuration file, then construct a GAR file containing the file and a `coherence-application.xml` file. The GAR file has the following structure:

   ```
   my.gar
   session-cache-config.xml
   META-INF
     coherence-application.xml
     MANIFEST.MF
   ```

   See Packaging Coherence Applications for WebLogic Server in *Administering Oracle Coherence* and Creating Coherence Applications for Oracle WebLogic Server in *Developing Oracle Coherence Applications for Oracle WebLogic Server*.

   a. Create the custom session cache configuration file and name it `session-cache-config.xml`.

Copyright © 2008, 2026, Oracle and/or its affiliates.

If you are deploying a GAR file, set the `local-storage` parameter in the custom `session-cache-config.xml` file to `true`, to configure all caches to start with storage enabled, for example:

```
<local-storage>true</local-storage>
```

> **Note**
>
> The `local-storage` parameter specifies whether a cluster node contributes storage to the cluster. In WebLogic Server, the `local-storage` parameter does not enable storage in Coherence*Web for WebLogic Server members that have a GAR file deployed to them.

b. Create a `coherence-application.xml` file. In the file, use the `cache-configuration-ref` parameter to reference your custom `session-cache-config.xml` file, for example:

```
<?xml version="1.0"?>
<coherence-application
   xmlns="http://xmlns.oracle.com/coherence/coherence-application">
<cache-configuration-ref>session-cache-config.xml</cache-configuration-
ref>
</coherence-application>
```

2. Deploy the GAR file to the WebLogic Server cluster that is to act as the storage-enabled Coherence cluster members.

   Note that storage must be enabled in either of the following ways:

   - Enable storage in the `session-cache-config.xml` file (see Step 1a).
   - Enable storage in the server itself either by enabling the **Coherence Web Local Storage Enabled** option. In the WebLogic Remote Console, go to the Edit Tree, then Environment, then Clusters, then *myCluster*, then the Coherence tab. At the command line, set the JVM argument `coherence.session.localstorage` to `true`.

   See Deploying Coherence Applications To a WebLogic Server Domain in *Administering Oracle Coherence* and Deploying Coherence Applications in WebLogic Server in *Developing Oracle Coherence Applications for Oracle WebLogic Server*.

Scoping the Session Cookie Path

WebLogic Server and Coherence*Web handle session scoping and the session lifecycle in different ways. This can impact your decision to implement a single sign-on (SSO) strategy for your applications.

By default, WebLogic Server uses the same session ID in every Web application for a given client, and sets the session cookie path to a forward slash (/). This is a requirement of the WebLogic Server default *thin* SSO implementation, which is enabled by default. By generating a session cookie with a path of "/", clients always return the same session ID in every request to the server. In WebLogic Server, a single session ID can be mapped to multiple session objects. Each Web application will have a different session object instance even though the session ID is identical (unless session sharing is enabled).

In contrast, Coherence*Web maps a session ID to a single session instance. This means that the behavior of having multiple session instances mapped to the same ID is not replicated by default if an application uses Coherence*Web. Because the session cookie is mapped to "/" by default, a single Coherence*Web session is shared across all Web applications. The default configuration in Coherence*Web is that all session attributes are scoped to a Web application.

Copyright © 2008, 2026, Oracle and/or its affiliates.

For most purposes, this single session approach is transparent. The major difference of having a single session across all Web applications is the impact of session invalidation. If Coherence*Web is enabled and you invalidate a session in one Web application, then you invalidate that session in all Web applications that use that session instance. If your Web applications do not use *thin* SSO, then you can avoid this issue by scoping the session cookie to the Web application path.

Therefore, you have the following options regarding SSO:

- Enable WebLogic Server session compatibly mode. This configuration is set with the `coherence-session-weblogic-compatibility-mode` parameter and mirrors all of the native WebLogic Server session persistence types: memory (single-server, non-replicated), file system persistence, JDBC persistence, cookie-based session persistence, and in-memory replication (across a cluster). By default, this mode is enabled. See Using Sessions and Session Persistence in *Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server*.
- Enable *thin* SSO functionality. Clients will use a single session across all Web applications. This means that the session life cycle will be inconsistent with all other session persistence types.
- Disable the *thin* SSO functionality by scoping the session cookie path to the Web application context path. This will allow the session life cycle to be consistent with all other session persistence types.

One advantage of enabling thin SSO with Coherence*Web is that it will work across all Web applications that are using the same Coherence cluster for Coherence*Web. The Coherence cluster is completely independent from the WebLogic Server cluster. The thin SSO functionality can even span multiple domains by enabling cross-domain trust in the WebLogic Server security layer.

Application requirements

Cookie path/domain and session cache scoping solve different problems. Use the narrowest intended cookie scope and keep authentication/authorization checks in the application. Confirm HTTPS/Secure, HttpOnly and SameSite settings in the selected container; Coherence session affinity is routing information, not an authentication token.

Updating the Session ID

When a user successfully authenticates a protected resource, the session ID is changed for security purposes.

In previous releases of WebLogic Server, a new session would be created, all of the session attributes from the old session would be copied into the new session, and then the old session would be invalidated. This would trigger the session listeners (if any were registered), so session lifecycle and session attribute listeners would be executed.

WebLogic Server supports the `HttpServletRequest.changeSessionId` method, introduced in Servlet 3.1 and retained in Jakarta Servlet. The implementation of the `changeSessionId` method allows the actual session ID to be updated. Changing only the ID does not recreate the session or copy its attributes, so it does not cause session creation/destruction or attribute-change notifications for that reason. A registered HttpSessionIdListener can receive the session-ID change notification. Most users should not notice any changes in the behavior of their applications.

For more information on the `HttpServletRequest.changeSessionId` method, see the Jakarta Servlet API and HttpSessionIdListener documentation:

Copyright © 2008, 2026, Oracle and/or its affiliates.

HttpSessionIdListener

Sharing Coherence*Web Sessions with Other Application Servers

If you are running Coherence*Web on WebLogic Server and on other application servers within a single cluster, then the session cookies created by WebLogic Server will not be decoded correctly by Coherence*Web on the other servers. This is because WebLogic Server adds a session affinity suffix to the cookie which is not part of the session ID stored in Coherence*Web. The other application servers must remove the WebLogic session affinity suffix from the session cookie value for Coherence*Web to be able to retrieve the session from the Coherence cache.

To strip the WebLogic session affinity suffix from the session cookie, add the `coherence-session-affinity-token` context parameter to the `web.xml` file used in the other application servers. Set the parameter value to an exclamation point (`!`), as illustrated in Example 2-2. The session affinity suffix will be removed from the session cookie when it is processed by the other application server.

Example 2-2 Removing Session Affinity Suffix

```
...
<context-param>
  <param-name>coherence-session-affinity-token</param-name>
  <param-value>!</param-value>
</context-param>
...
```

See Coherence*Web Context Parameters for more information on the `coherence-session-affinity-token` context parameter.

Copyright © 2008, 2026, Oracle and/or its affiliates.

3

Using Coherence*Web on Other Application Servers

You can configure and deploy Coherence*Web, the session state persistence and management module, for use with a variety of application servers. The functionality that allows Coherence*Web to be used with these application servers is provided by running the automated Coherence*Web WebInstaller.

> **Note**
>
> Consult Supported Web Containers to see if you must perform any application server-specific integration steps.

This chapter includes the following sections:

- Integrating Coherence*Web Using the WebInstaller
- Coherence*Web WebInstaller Ant Task
- Testing HTTP Session Management
- How the Coherence*Web WebInstaller Instruments a Jakarta EE Application
  Coherence*Web WebInstaller performs many tasks as a part of the inspection step.
- Integrating Coherence*Web with Applications Using Jakarta Security
  Coherence*Web can be integrated with applications that use Jakarta Security.
- Preventing Cross-Site Scripting Attacks
  Use the `coherence-session-cookie-httponly` context parameter to append the `HttpOnly` attribute to the session cookie. The `HttpOnly` attribute limits script access to the cookie. It helps protect the cookie value if script injection occurs, but does not prevent cross-site scripting or authorize a request.
- Selecting CoherenceWeb Artifacts and Container APIs
  Match the module, Servlet API, class loader and release before installing or instrumenting an application.

Integrating Coherence*Web Using the WebInstaller

Coherence*Web can be enabled for Jakarta EE applications on several different Web containers.
To enable the Coherence*Web, you must run the ready-to-deploy application through the automated Coherence*Web WebInstaller before deploying it. This utility prepares the application for deployment. It performs the integration process in two discrete steps: an inspection step and an integration step. See How the Coherence*Web WebInstaller Instruments a Jakarta EE Application.

WebInstaller can be run either from the Java command line or from Ant tasks. The following sections describe the Java command-line method. For an Ant task-based environment, see Coherence*Web WebInstaller Ant Task.

Copyright © 2008, 2026, Oracle and/or its affiliates.

This section includes the following topics:

- General Instructions for Integrating Coherence*Web Session Management Module
- Deploying and Running Applications In Process
- Deploying and Running Applications Out-of-Process
- Migrating to Out-of-Process Topology
- Deploying and Running Applications Out-of-Process with Coherence*Extend
- Enabling Sticky Sessions for Apache Tomcat Servers
- Integrating with IBM WebSphere Liberty

General Instructions for Integrating Coherence*Web Session Management Module

Complete the following steps to integrate Coherence*Web with a Jakarta EE application on any of the Web containers listed under Supported Web Containers.

If you are integrating Coherence*Web with a Jakarta EE application on an Apache Tomcat Server, see also Enabling Sticky Sessions for Apache Tomcat Servers for additional instructions.

If you are integrating Coherence*Web with a Jakarta EE application on an IBM WebSphere Server, see also Integrating with IBM WebSphere Liberty for additional instructions.

To integrate Coherence*Web for the Jakarta EE application you are deploying:

1. Ensure that the application directory and the EAR file or WAR file are not being used or accessed by another process.
2. Change the current directory to the Coherence library directory (`%COHERENCE_HOME%\lib` on Windows and `$COHERENCE_HOME/lib` on UNIX).
3. Ensure that the paths are configured so that Java commands will run.
4. Complete the application inspection step by running the following command. Specify the full path to your application and the name of your server found in Table 1-1 (replacing the `<app-path>` and `<server-type>` with them in the following command line):

   ```
   java -jar webInstaller.jar <app-path> -inspect -server:<server-type>
   ```

   The system will create (or update, if it already exists) the `coherence-web.xml` configuration descriptor file for your Jakarta EE application in the directory where the application is located. This configuration descriptor file contains the default Coherence*Web settings for your application as recommended by the utility.
5. If necessary, review and modify the Coherence*Web settings based on your requirements.

   You can modify the Coherence*Web settings by editing the `coherence-web.xml` descriptor file. Coherence*Web Context Parameters, describes the Coherence*Web settings that can be modified. Use the `param-name` and `param-value` subelements of the `context-param` parameter to enable the features you want. Table 3-1 describes some examples of different settings.

Copyright © 2008, 2026, Oracle and/or its affiliates.

Table 3-1 Example Context Parameter Settings for Coherence*Web

Parameter	Name	Description
`coherence-servletcontext-clustered`	true	Clusters all `ServletContext` (global) attributes so that servers in a cluster share the same values for those attributes, and also receive the events specified by the Servlet Specification when those attributes change. **Note** This property is not applicable for IBM WebSphere Liberty.
`coherence-enable-sessioncontext`	true	Allows an application to enumerate all of the sessions that exist within the application, or to obtain any one of those sessions to examine or manipulate.
`coherence-session-id-length`	32	Enables you to increase the length of the `HttpSession` ID, which is generated using a `SecureRandom` algorithm; the length can be any value, although in practice it should be small enough to fit into a cookie or a URL (depending on how session IDs are maintained.) Increasing the length can decrease the chance of a session being purposely hijacked.
`coherence-session-urlencode-enabled`	true	By default, the `HttpSession` ID is managed in a cookie. If the application supports URL encoding, this option enables it.

6. Complete the Coherence*Web application integration step by running the following command, replacing `<app-path>` with the full path to your application:

   ```
   java -jar webInstaller.jar <app-path> -install
   ```

   The `coherence-web.jar` file that gets added, includes the `default-session-cache-config.xml` file that contains the session and cache configuration information.

7. Deploy the updated application and verify that everything functions as expected, using the lightweight load balancer provided with the Coherence distribution. Remember that the lightweight load balancer is not a production-ready utility, in contrast to the load balancer provided by WebLogic Server.

   The application can be deployed and run in any of the deployment topologies supported by Coherence: in-process, out-of-process, or out-of-process with Coherence*Extend. See the following sections for information on deploying and running your applications under these topologies. See Deployment Topologies.

Deploying and Running Applications In Process

Coherence*Web can be run *in-process* with the application server. This is where session data is stored with the application server. See In-Process Topology for more information on this topology.

For the application server:

1. Start the application server in storage-enabled mode. Add the system property `coherence.session.localstorage=true` to the Java options of your application server startup script.
2. Deploy the `coherence.jar` and `coherence-web.jar` files as shared libraries.

Copyright © 2008, 2026, Oracle and/or its affiliates.

3. Deploy and run your application.

Deploying and Running Applications Out-of-Process

In the out-of-process deployment topology, a stand-alone cache server stores the session data and the application server is configured as a cache client. See Out-of-Process Topology.

The cache server and the application server must use the same cache and session configuration. This configuration is generated in the `default-session-cache-config.xml` file by the Coherence*Web WebInstaller. The WebInstaller generates the file in the `WEB-INF\classes` directory of the instrumented application.

For the cache server:

1. Add the `coherence.cacheconfig` system property to the cache server startup script to locate the file configuration file. You must also include the system property `coherence.session.localstorage=true` to enable storage for the cache server.
2. Add the `coherence.jar` and `coherence-web.jar` files to the classpath in the cache server startup script.

   Following is a sample startup script:

   ```
   java -server -Xms512m -Xmx512m
   -cp <Coherence installation dir>/lib/coherence.jar:<Coherence installation
   dir>/lib/coherence-web.jar -Dcoherence.management.remote=true -
   Dcoherence.cacheconfig=default-session-cache-config.xml
   -Dcoherence.session.localstorage=true com.tangosol.net.DefaultCacheServer
   ```

For the application server (cache client):

1. Deploy the `coherence.jar` and `coherence-web.jar` files as shared libraries.
2. The `default-session-cache-config.xml` file should already be present in the `WEB-INF\classes` directory of the instrumented application.

   By default, the file should specify that local storage is disabled (if you are not sure, you can either inspect the file to confirm that the `local-storage` element is set to `false` or add the system property `coherence.session.localstorage=false` to the startup script).
3. Deploy the application to the server.

Migrating to Out-of-Process Topology

If you have been running and testing your application with Coherence*Web in-process, you can easily migrate to the out-of-process topology. Simply set up your cache server and application server as described in Deploying and Running Applications Out-of-Process.

Deploying and Running Applications Out-of-Process with Coherence*Extend

The out-of-process with Coherence*Extend topology is similar to the out-of-process topology except that the communication between the application server tier and the cache server tier is over Coherence*Extend (TCP/IP). Coherence*Extend consists of two components: an extend client (or *proxy*) running outside the cluster and an extend proxy service running in the cluster hosted by one or more cache servers. See Out-of-Process with Coherence*Extend Topology.

In these deployments, there are three types of participants:

Copyright © 2008, 2026, Oracle and/or its affiliates.

- Cache servers (storage servers), which are used to store the actual session data in memory.
- Web (application) servers, which are the Extend clients in this topology. They are not members of the cluster; instead, they connect to a proxy node in the cluster that will issue requests to the cluster on their behalf.
- Proxy servers, which are storage-disabled members (nodes) of the cluster that accept and manage TCP/IP connections from Extend clients. Requests that arrive from clients will be sent into the cluster, and responses will be sent back through the TCP/IP connections.

For the cache server:

Follow the instructions for configuring the cache server in Deploying and Running Applications Out-of-Process. Also, edit the cache server's copy of the `default-session-cache-config.xml` file to add the system properties `coherence.session.proxy=false` and `coherence.session.localstorage=true`.

See Configure the Cache for Proxy and Storage JVMs for more information and an example of a `default-session-cache-config.xml` file with these context parameters.

For the Web tier (application) server:

Follow the instructions for configuring the application server in Deploying and Running Applications Out-of-Process. Also, complete these steps:

1. Ensure that Coherence*Web is configured to use the Optimistic Locking mode. Optimistic locking is the default locking mechanism for Coherence*Web (see Optimistic Locking).
2. Edit the application server's copy of the `default-session-cache-config.xml` file to add the proxy JVM host names, IP addresses and ports. To do this, add a `<remote-addresses>` section to the file. In most cases, you should include the host name and IP address, and port of all proxy JVMs for load balancing and failover.

See Configure the Cache for Web Tier JVMs.

For the proxy server:

With a few changes, the proxy server can use the same cache and session configuration as the application server and the cache server. Edit the `default-session-cache-config.xml` file to add these system properties:

- `coherence.session.localstorage=false` to disable local storage.
- `coherence.session.proxy=true` to indicate that a proxy service is being used.
- `coherence.session.proxy.localhost` to indicate the host name or IP address of the NIC to which the proxy will bind.
- `coherence.session.proxy.localport` to indicate a unique port number to which the proxy will bind.

See Configure the Cache for Proxy and Storage JVMs.

Enabling Sticky Sessions for Apache Tomcat Servers

If you want to employ sticky sessions for the Apache Tomcat Server, you must configure the `jvmRoute` attribute in the server's `server.xml` file. You can find more information on this attribute at this URL:

`http://tomcat.apache.org/connectors-doc/reference/workers.html`

Copyright © 2008, 2026, Oracle and/or its affiliates.

Integrating with IBM WebSphere Liberty

HTTP session affinity may need to be explicitly configured when integrating with WebSphere Liberty. Coherence*Web needs to be passed the Clone ID of the Liberty server as well as the affinity separator. If the Clone ID is defined by the user, as explained in the liberty documentation at https://www.ibm.com/support/knowledgecenter/SSEQTP_8.5.5/com.ibm.websphere.wlp.doc/ae/twlp_admin_session_persistence.html, and if the affinity separator is the colon character (`:`) character, then no additional configuration is required. If that is not the case, then the following system properties can be used during server startup:

- `coherence.web.liberty.suffix.separator` – The affinity suffix separator. The default value is `:`.
- `coherence.web.liberty.suffix` – The clone id of the server. The default value is the value configured for the `cloneId` system property in the `bootstrap.properties` file as explained in the WebSphere Liberty documentation cited above.

Coherence*Web WebInstaller Ant Task

The Coherence*Web WebInstaller Ant task enables you to run the utility from within your existing Ant build files.
This section includes the following topics:

- Using the Coherence*Web WebInstaller Ant Task
- Configuring the WebInstaller Ant Task
- WebInstaller Ant Task Examples

Using the Coherence*Web WebInstaller Ant Task

To use the Coherence*Web WebInstaller Ant task, add the task import statement illustrated below in to your Ant build file. In this example, `${coherence.home}` refers to the root directory of your Coherence installation.

```
<taskdef name="cwi" classname="com.tangosol.coherence.misc.CoherenceWebAntTask">
    <classpath>
        <pathelement location="${coherence.home}/lib/webInstaller.jar"/>
    </classpath>
</taskdef>
```

The following procedure describes the basic process of integrating Coherence*Web with a Jakarta EE application from an Ant build:

1. Build your Jakarta EE application as you ordinarily would.
2. Run the Coherence*Web Ant task with the `operations` attribute set to `inspect`.
3. Make any necessary changes to the generated Coherence*Web XML descriptor file.
4. Run the Coherence*Web Ant task with the `operations` attribute set to `install`.

Performing Iterative Development

If you are performing iterative development on your application, such as modifying Jakarta Server Pages (JSPs), Servlets, static resources, and so on, use the following integration process:

Copyright © 2008, 2026, Oracle and/or its affiliates.

1. Run the Coherence*Web Ant task with the `operations` attribute set to `uninstall`, the `failonerror` attribute set to `false`, and the `descriptor` attribute set to the location of the previously generated Coherence*Web XML descriptor file (from Step 2 of Using the Coherence*Web WebInstaller Ant Task).
2. Build your Jakarta EE application as you ordinarily would.
3. Run the Coherence*Web Ant task with the `operations` attribute set to `inspect`, and the `install` and `descriptor` attributes set to the location of the previously generated Coherence*Web XML descriptor file (from Step 2 of Using the Coherence*Web WebInstaller Ant Task).

Changing the Coherence*Web Configuration Settings of a Jakarta EE Application

If you must change the Coherence*Web configuration settings of a Jakarta EE application that is using Coherence*Web, follow these steps:

1. Run the Coherence*Web Ant task with the `operations` attribute set to `uninstall` and the `descriptor` attribute set to the location of the Coherence*Web XML descriptor file for the Jakarta EE application.
2. Change the necessary configuration parameters in the Coherence*Web XML descriptor file.
3. Run the Coherence*Web Ant task with the `operations` attribute set to `install` and the `descriptor` attribute set to the location of the modified Coherence*Web XML descriptor file (from Step 2 of Using the Coherence*Web WebInstaller Ant Task).

Configuring the WebInstaller Ant Task

Table 3-2 describes the attributes that can be used with the Coherence*Web WebInstaller Ant task.

Table 3-2 Coherence*Web WebInstaller Ant Task Attributes

Attribute	Description	Required?
`app`	Path to the target Jakarta EE application. This can be a path to a WAR file, an EAR file, an expanded WAR directory, or an expanded EAR directory.	Yes, if the `operations` attribute is set to any value other than `version`.
`backup`	Path to a directory that holds a backup of the original target Jakarta EE application. This attribute defaults to the directory that contains the Jakarta EE application.	No
`descriptor`	Path to the Coherence*Web XML descriptor file. This attribute defaults to the `coherence-web.xml` file in the directory that contains the target Jakarta EE application.	No
`failonerror`	Stops the Ant build if the Coherence*Web WebInstaller exits with a status other than 0. The default is `true`.	No
`nowarn`	Suppresses warning messages. This attribute can be either `true` or `false`. The default is `false`.	No
`operations`	A comma- or space-separated list of operations to perform; each operation must be one of `inspect`, `install`, `uninstall`, or `version`.	Yes
`server`	The alias of the target Jakarta EE application server.	No

Copyright © 2008, 2026, Oracle and/or its affiliates.

Table 3-2 (Cont.) Coherence*Web WebInstaller Ant Task Attributes

Attribute	Description	Required?
`touch`	Touches JSPs and TLDs that are modified by the Coherence*Web WebInstaller. This attribute can be either `true`, `false`, or `M/d/y h:mm a'` The default is `false`.	No
`verbose`	Displays verbose output. This attribute can be either `true` or `false`. The default is `false`.	No

WebInstaller Ant Task Examples

The following list provides sample commands for the WebInstaller Ant task.

- Inspect the `myWebApp.war` Web application and generate a Coherence*Web XML descriptor file called `my-coherence-web.xml` in the current working directory:

  ```
  <cwi app="myWebApp.war" operations="inspect" descriptor="my-coherence-web.xml"/>
  ```

- Integrate Coherence*Web into the `myWebApp.war` Web application using the Coherence*Web XML descriptor file called `my-coherence-web.xml` found in the current working directory:

  ```
  <cwi app="myWebApp.war" operations="install" descriptor="my-coherence-web.xml"/>
  ```

- Remove Coherence*Web from the `myWebApp.war` Web application:

  ```
  <cwi app="myWebApp.war" operations="uninstall"/>
  ```

- Integrate Coherence*Web into the `myWebApp.war` Web application located in the `/dev/myWebApp/build` directory using the Coherence*Web XML descriptor file called `my-coherence-web.xml` found in the `/dev/myWebApp/src` directory, and place a backup of the original Web application in the `/dev/myWebApp/work` directory:

  ```
  <cwi app="/dev/myWebApp/build/myWebApp.war" operations="install" descriptor="/dev/
  myWebApp/src/my-coherence-web.xml" backup="/dev/myWebApp/work"/>
  ```

- Integrate Coherence*Web into the `myWebApp.war` Web application located in the `/dev/myWebApp/build` directory using the Coherence*Web XML descriptor file called `coherence-web.xml` found in the `/dev/myWebApp/build` directory. If the Web application has not already been inspected (that is, `/dev/myWebApp/build/coherence-web.xml` does not exists); inspect the Web application before integrating Coherence*Web:

  ```
  <cwi app="/dev/myWebApp/build/myWebApp.war" operations="inspect,install"/>
  ```

- Reintegrate Coherence*Web into the `myWebApp.war` Web application located in the `/dev/myWebApp/build` directory, using the Coherence*Web XML descriptor file called `my-coherence-web.xml` found in the `/dev/myWebApp/src` directory:

  ```
  <cwi app="/dev/myWebApp/build/myWebApp.war" operations="uninstall,install"
  descriptor="/dev/myWebApp/src/my-coherence-web.xml"/>
  ```

Testing HTTP Session Management

Coherence comes with a lightweight software load balancer intended only for testing purposes. The load balancer is very easy to use and is very useful when testing functionality such as session management.
Follow these steps to test HTTP session management with the lightweight load balancer:

Copyright © 2008, 2026, Oracle and/or its affiliates.

1. Start multiple application server processes on one or more server machines, each running your application on a unique IP address and port combination.
2. Open a command (or shell) window.
3. Change the current directory to the Coherence library directory (`%COHERENCE_HOME%\lib` on Windows and `$COHERENCE_HOME/lib` on UNIX).
4. Ensure that paths are configured so that Java commands will run.
5. Start the software load balancer with the following command lines (each of these command lines makes the application available on the default HTTP port `80`).

   For example, to test load balancing locally on one machine with two application server instances on ports `7001` and `7002`:

   ```
   java -jar coherence-loadbalancer.jar localhost:80 localhost:7001 localhost:7002
   ```

   To run the load balancer locally on a machine named `server1` that load balances to port `7001` on `server1`, `server2`, and `server3`:

   ```
   java -jar coherence-loadbalancer.jar server1:80 server1:7001 server2:7001
   server3:7001
   ```

   Assuming that you use the preceding command line, an application that previously was accessed with the URL `http://server1:7001/my.jsp` would now be accessed with the URL `http://server1:80/my.jsp` or just `http://server1/my.jsp`.

> **Note**
>
> Ensure that your application uses only relative redirections or the address of the load balancer.

Table 3-3 describes the command-line options for the load balancer:

Table 3-3 Load Balancer Command-Line Options

Option	Description
`backlog`	Sets the TCP/ IP accept backlog option to the specified value, for example: `-backlog=64`.
`random`	Specifies the use of a random load-balancing algorithm (default).
`roundrobin`	Specifies the use of a round-robin load-balancing algorithm.
`threads`	Uses the specified number of request or response thread pairs (so the total number of additional daemon threads will be two times the specified value), for example: `-threads=64`.

How the Coherence*Web WebInstaller Instruments a Jakarta EE Application

Coherence*Web WebInstaller performs many tasks as a part of the inspection step.

The following are the tasks:

Copyright © 2008, 2026, Oracle and/or its affiliates.

1. Generates a template `coherence-web.xml` configuration file that contains basic information about the application and target Web container along with a set of default Coherence*Web configuration context parameters appropriate for the target Web container. See Coherence*Web Context Parameters.

   The WebInstaller sets the servlet container to start in storage-disabled mode (that is, it sets `coherence.session.localstorage` to `false`).

   If an existing `coherence-web.xml` configuration file exists (for example, from a previous run of the Coherence*Web WebInstaller), the context parameters in the existing file are merged with those in the generated template.
2. Enumerates the JSP from each Web application in the target Jakarta EE application and adds information about each JSP to the `coherence-web.xml` configuration file.
3. Enumerates the TLDs from each Web application in the target Jakarta EE application and adds information about each TLD to the `coherence-web.xml` configuration file.
4. Scans the web application and all classes contained within, for annotations and generates markup that is inserted into the appropriate `web.xml` or `web-fragment.xml`.
5. For each ServletContainerInitializer found, create a custom ServletContainerInitializer that will invoke the application's initializer when the context starts. This allows Coherence*Web to enact properly on dynamically registered Servlet artifacts.

During the integration step, the Coherence*Web WebInstaller performs the following tasks:

1. Creates a backup of the original Jakarta EE application so that it can be restored during the uninstallation step.
2. Adds the Coherence*Web configuration context parameters generated in Step 1 of the inspection step to the `web.xml` descriptor file of each Web application contained in the target Jakarta EE application.
3. Unregisters any application-specific `ServletContextListener`, `ServletContextAttributeListener`, `ServletRequestListener`, `ServletRequestAttributeListener`, `HttpSessionListener`, and `HttpSessionAttributeListener` classes (including those registered by TLDs) from each Web application.
4. Registers a Coherence*Web `ServletContextListener` class in each `web.xml` descriptor file. At run time, the Coherence*Web `ServletContextListener` class propagates each `ServletContextEvent` event to each application-specific `ServletContextListener` listener.
5. Registers a Coherence*Web `ServletContextAttributeListener` listener in each `web.xml` descriptor file. At run time, the Coherence*Web `ServletContextAttributeListener` propagates each `ServletContextAttributeEvent` event to each application-specific `ServletContextAttributeListener` listener.
6. Wraps each application-specific `Servlet` declared in each `web.xml` descriptor file with a Coherence*Web `SessionServlet`. At run time, each Coherence*Web `SessionServlet` delegates to the wrapped `Servlet`.
7. Adds the following directive to each JSP enumerated in Step 2 of the inspection step:

   ```
   <%@ page extends="com.tangosol.coherence.servlet.api22.JspServlet" %>
   ```


Copyright © 2008, 2026, Oracle and/or its affiliates.

> **Note**
>
> The presence of duplicate page extends in JSP may cause errors after Coherence*Web is integrated with the Jakarta EE application. Therefore, remove the replicated page extends. You do not need to add page extends explicitly because the Coherence*Web WebInstaller utility adds page extends for the Coherence servlet context by default.

During the uninstallation step, the Coherence*Web WebInstaller replaces the instrumented Jakarta EE application with the backup of the original version created in Step (1) of the integration process.

Integrating Coherence*Web with Applications Using Jakarta Security

Coherence*Web can be integrated with applications that use Jakarta Security.

To integrate Coherence*Web with an application that uses Jakarta Security, follow these additional steps:

1. Enable Coherence*Web session cookies.

   See the `coherence-session-cookies-enabled` configuration element in Table A-1 for additional details.

2. Change the Coherence*Web session cookie name to a name that is different from the one used by the target Web container.

   By default, most containers use `JSESSIONID` for the session cookie name, so a good choice for the Coherence*Web session cookie name is `CSESSIONID`. See the `coherence-session-cookie-name` configuration element in Table A-1 for additional details.

3. Enable session replication for the target Web container.

   If session replication is not enabled, or the container does not support a form of session replication, then you will be forced to re-authenticate to the Web application during failover. See your Web container's documentation for instructions on enabling session replication.

This configuration causes two sessions to be associated with a given authenticated user:

- A Coherence*Web session that contains all session data created by the Web application
- A session created by the Web container during authentication that stores only information necessary to identify the user

Preventing Cross-Site Scripting Attacks

Use the `coherence-session-cookie-httponly` context parameter to append the `HttpOnly` attribute to the session cookie. The `HttpOnly` attribute limits script access to the cookie. It helps protect the cookie value if script injection occurs, but does not prevent cross-site scripting or authorize a request.

Configure HTTPS and the Secure cookie attribute, appropriate SameSite/path/domain policy and application output encoding/CSRF protection through the container/application. This context parameter is available for instrumented applications only.

Copyright © 2008, 2026, Oracle and/or its affiliates.

Selecting CoherenceWeb Artifacts and Container APIs

Match the module, Servlet API, class loader and release before installing or instrumenting an application.

Choose the Coherence*Web module for the actual container Servlet API and the matching Coherence distribution. All modules in the deployment must use compatible release/patch versions; do not combine artifacts just because both use jakarta packages.

Table 3-4 Jakarta Coherence*Web module selection

Module	Platform/API baseline	Placement requirement
`coherence-web-ee9`	Jakarta EE 9 / Servlet 5.0	Use the documented container integration and keep one compatible servlet API supplied by the container.
`coherence-web-ee10`	Jakarta EE 10 / Servlet 6.0	Use the matching module/ dependencies; do not place a duplicate servlet implementation in the application.
`coherence-web-ee11`	Jakarta EE 11 / Servlet 6.1	Use the matching container/ integration; a Java baseline alone does not certify a container version.

This table describes the module API targets, not a blanket certification of every container at those levels. WebLogic uses its integrated distribution and supported domain patch procedure. For other servers, follow the selected container instructions, WebInstaller alias or native Tomcat integration and the actual artifact layout.

- Record server/container, Java, Coherence and module versions before deployment. Confirm the supported combination using the distribution certification/release information.
- Keep application session attribute classes, serializers/POF resources and generated trust metadata visible to all session-deserializing tiers.
- Preserve an original WAR/EAR and inspect the instrumented output before deployment. Do not repeatedly instrument an already modified archive without the documented uninstall/reinstall workflow.
- Test create/read/update/invalidate, authentication ID change, failover and restart with representative attributes and listeners. Check the actual class loader on both application and storage tiers.

Use Preparing a Release-Matched WebLogic Deployment for WebLogic and Upgrading and Operating a Secured Deployment for first-start checks.

Copyright © 2008, 2026, Oracle and/or its affiliates.

4

Tomcat Native Session Integration

If you are using Tomcat 10.0.x (Servlet 5.0), Tomcat 10.1.x (Servlet 6.0) or Tomcat 11.0.x (Servlet 6.1) , then you may opt to use Coherence*Web's native session integration instead of using the installer, simplifying the development and deployment process of Coherence*Web applications.

This chapter includes the following sections:

Application requirements

Install the manager/valve and dependencies at the class-loader level required by the selected Tomcat integration. Keep only one compatible Servlet API and Coherence module set. Verify undeploy/redeploy cleanup and class-loader release; do not copy WebInstaller-specific parameters into native manager configuration without checking their support.

For current preparation use Selecting CoherenceWeb Artifacts and Container APIs and Preparing Session Attributes for Hardened Mode.

- Tomcat Configuration
- Manager Configuration Options

Tomcat Configuration

Before deploying a Coherence*Web application to Tomcat, first copy the `coherence.jar` and `coherence-web.jar` files to `CATALINA_HOME/lib` on *nix or `%TOMCAT_HOME%\lib` on Windows. The JARs must not be included with the web application.

Each web application that will be using Coherence*Web will need to include a Tomcat-specific configuration file, `META-INF/context.xml`. The file must minimally include the following:

```
<Context>
    <Manager className="com.tangosol.coherence.web.tomcat.Manager"/>
    <Valve className="com.tangosol.coherence.web.tomcat.RequestValve"/>
</Context>
```

Those familiar with previous releases of Coherence*Web are familiar with configuring Coherence*Web using context initialization parameters. When using the Tomcat native session integration, these context attributes are not needed, and if present, will be ignored. Instead, Coherence*Web is configured using attributes on the Manager element from the preceding example. It should be noted that the available configuration options are a subset of those that are available when using the installer, as the native session integration's scope is much narrower. For example, when using the installer, there are various options to configure how sessions are handled, such as `http-only` cookies, the name of the session cookie, and such. Those configuration options, in the native integration case, are configured and handled by Tomcat and not by Coherence*Web.

Application requirements

Before deployment, complete Selecting CoherenceWeb Artifacts and Container APIs. Container API level, native integration and distribution certification are separate checks. Keep the container-provided Servlet API and lifecycle ownership intact.

Copyright © 2008, 2026, Oracle and/or its affiliates.

Manager Configuration Options

In addition to the standard manager options documented by Tomcat, the following configuration options are available when using the native session integration. See Table 4-1.

Table 4-1 Coherence Manager Configuration Options

Attribute Name	Description
`applicationName`	See coherence-application-name.
`configurationConsistency`	See coherence-configuration-consistency.
`collectionClass`	See coherence-sessioncollection-class.
`coordinatedReaper`	See coherence-reaperdaemon-cluster-coordinated.
`reaperSweepModulo`	See coherence-reaperdaemon-sweep-modulo.
`reaperLocality`	See coherence-reaperdaemon-assume-locality.
`reaperCycleSeconds`	See coherence-reaperdaemon-cycle-seconds.
`reaperPriority`	See coherence-reaperdaemon-priority.
`readInParallel`	See coherence-reaperdaemon-parallel.
`reapThreadsMin`	See coherence-reaperdaemon-min-threads.
`reapThreadsMax`	See coherence-reaperdaemon-max-threads.
`reapMechanism`	See coherence-session-reaping-mechanism.
`sessionCache`	See coherence-session-cachename.
`managementCache`	See coherence-session-management-cachename.
`sessionLocking`	See coherence-session-locking.
`memberLocking`	See coherence-session-member-locking.
`applicationLocking`	See coherence-session-app-locking.
`threadLocking`	See coherence-session-thread-locking.
`sessionGetLockTimeout`	See coherence-session-get-lock-timeout.
`logThreads`	See coherence-session-log-threads-holding-lock.
`logLevel`	See coherence-session-logger-level.
`scopeControllerClass`	See coherence-scopecontroller-class.
`overflowCache`	See coherence-session-overflow-cachename.
`overflowThreshold`	See coherence-attribute-overflow-threshold.
`suspectAttributes`	See coherence-enable-suspect-attributes.
`distributionControllerClass`	See coherence-distributioncontroller-class.
`preserveAttributes`	See coherence-preserve-attributes.
`localSessionCache`	See coherence-local-session-cachename.
`localAttributeCache`	See coherence-local-attribute-cachename.
`configurationPath`	See coherence-cache-configuration-path.
`federated`	See coherence-session-cache-federated.
`logInvalidationExceptions`	See coherence-session-log-invalidation-exceptions.
`clusterOwned`	See coherence-cluster-owned.

Copyright © 2008, 2026, Oracle and/or its affiliates.

Table 4-1 (Cont.) Coherence Manager Configuration Options

Attribute Name	Description
`shutdownDelaySeconds`	See coherence-shutdown-delay-seconds.

Copyright © 2008, 2026, Oracle and/or its affiliates.

5

Coherence*Web Session Management Features

Coherence*Web includes many features such as session models, session scoping, session locking, deployment topologies, and logging. You can configure these features to meet the demands of your environment. Consequently, you might have to change some default configuration options. This chapter provides an in-depth look at the features that Coherence*Web supports so that you can make the appropriate configuration and deployment decisions.
This chapter includes the following sections:

- Session Models
  A session model describes how Coherence*Web stores the session state in Coherence.
- Session and Session Attribute Scoping
- Cluster Node Isolation
- Session Locking Modes
  Oracle Coherence provides different configuration options for concurrent access to HTTP sessions.
- Deployment Topologies
- Accessing Sessions with Lazy Acquisition
  Lazy acquisition can be enabled to avoid unnecessary acquiring of a session whenever a servlet or filter is called.
- Overriding the Distribution of HTTP Sessions and Attributes
- Detecting Changed Attribute Values
- Saving Non-Serializable Attributes Locally
- Securing Coherence*Web Deployments
- Customizing the Name of the Session Cache Configuration File
- Configuring Logging for Coherence*Web
- Getting Concurrent Access to the Same Session Instance
  A cache delegator is used for applications that require concurrent access to the same session instance.
- Federated Session Caches
- Preparing Session Attributes for Hardened Mode
  Package compatible session models and generated trust metadata on every receiving session tier.

Session Models

A session model describes how Coherence*Web stores the session state in Coherence.

This section includes the following topics:

- Overview of Session Models

Copyright © 2008, 2026, Oracle and/or its affiliates.

- Monolithic Model
- Traditional Model
- Split Model
- Session Model Recommendations
- Configuring a Session Model
- Sharing Data in a Clustered Environment
- Scalability and Performance

Overview of Session Models

Session data is managed by an `HttpSessionModel` object while the session collection in a Web application is managed by an `HttpSessionCollection` object. You must configure only the collection type in the `web.xml` file—the model is implicitly derived from the collection type. Coherence*Web includes these different session model implementations:

- Monolithic Model, which stores all session state as a single entity, serializing and deserializing all attributes as a single operation
- Traditional Model, which stores all session state as a single entity but serializes and deserializes attributes individually
- Split Model, which extends the Traditional Model, but separates the larger session attributes into independent physical entities

These sections provide additional information on session models:

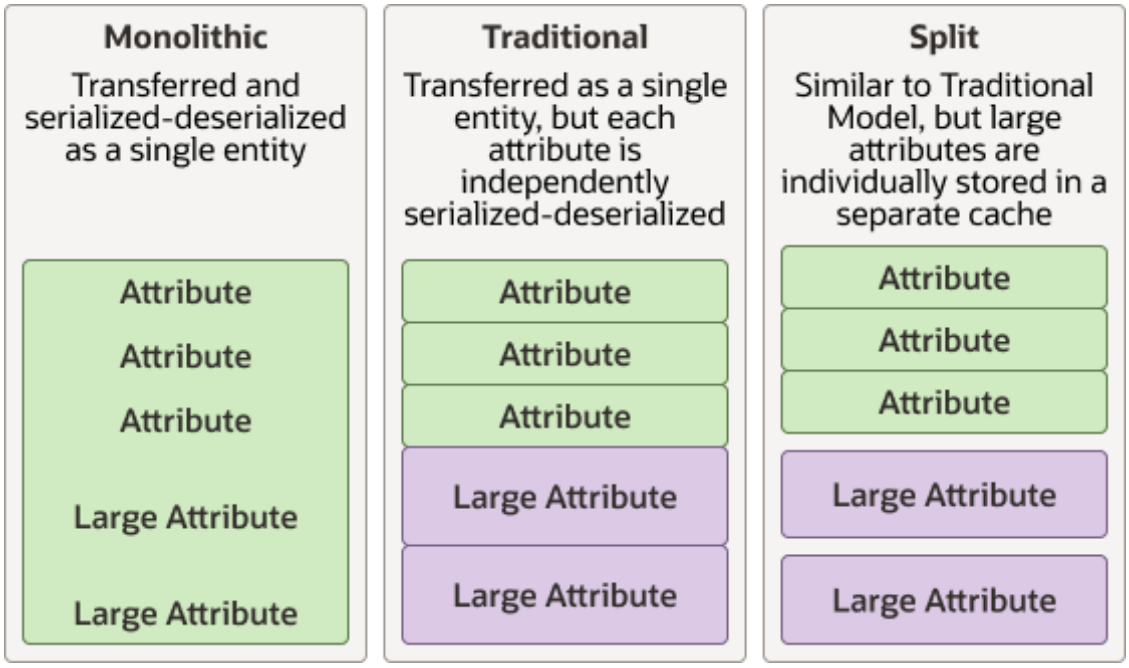
- Session Model Recommendations, provides recommendations on which session model to choose for your applications
- Configuring a Session Model, describes how to change the session model by using a system property or a context parameter
- Sharing Data in a Clustered Environment, describes how data is shared between and within JVMs
- Scalability and Performance. describes the impact of session models on scalability and performance

> **Note**
>
> In general, Web applications that are part of the same Coherence cluster must use the same session model type. Inconsistent configurations could result in deserialization errors.

Figure 5-1 illustrates the three session models.

Copyright © 2008, 2026, Oracle and/or its affiliates.

Figure 5-1 Traditional, Monolithic, and Split Session Models



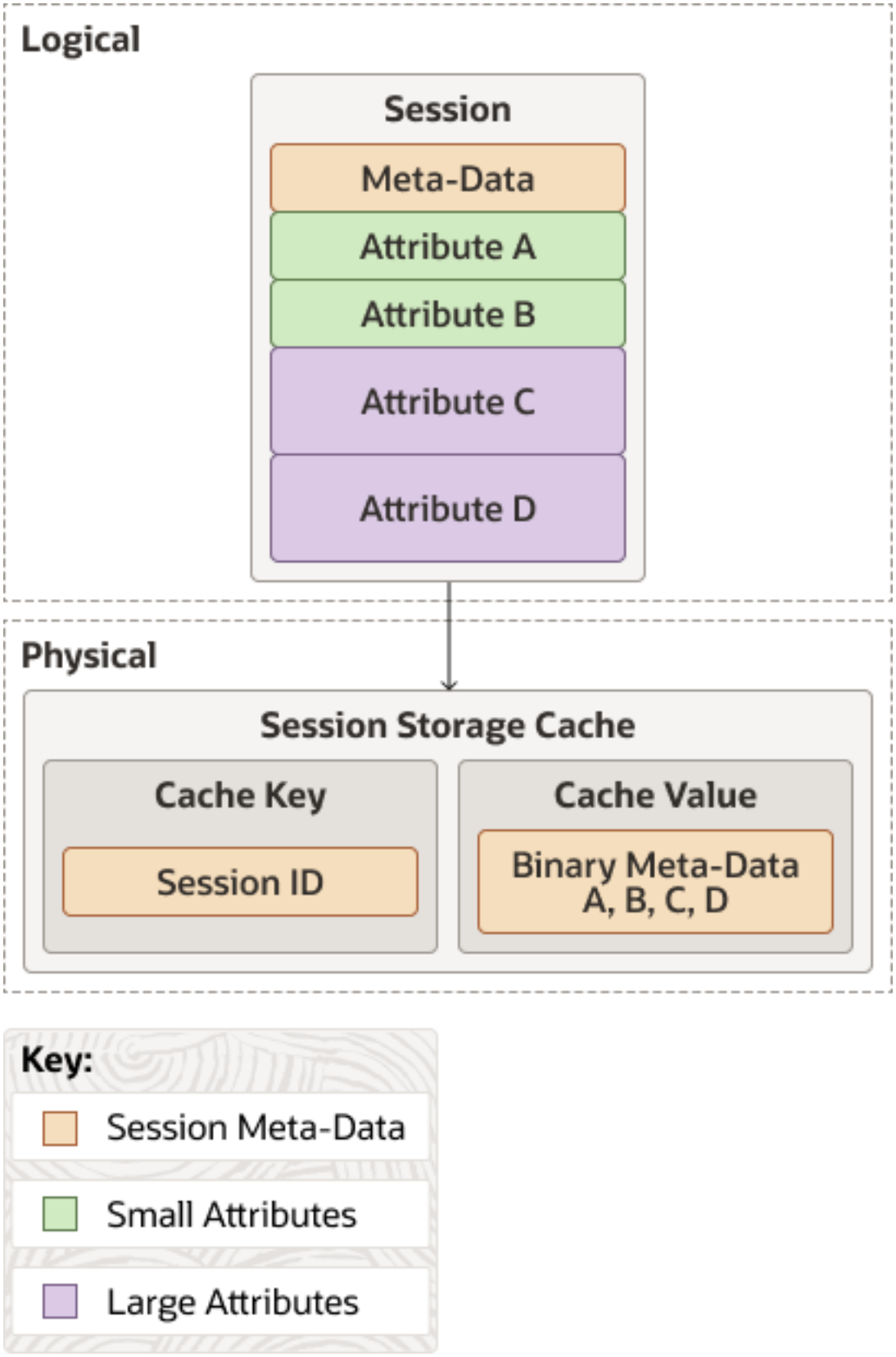
Monolithic Model

The Monolithic model is represented by the `MonolithicHttpSessionModel` and `MonolithicHttpSessionCollection` objects. These are similar to the Traditional model, except that they solve the shared object issue by serializing and deserializing all attributes into a single object stream. As a result, the Monolithic model often does not perform as well as the Traditional model.

Figure 5-2 illustrates the relationship between the logical representation of data and its physical representation in the session storage cache. In its logical representation session data consists of metadata, and various attributes. In its physical representation in the session storage cache, the metadata and attributes are serialized into a single stream. A session ID is associated with the metadata and attributes.

Copyright © 2008, 2026, Oracle and/or its affiliates.

Figure 5-2 Monolithic Session Model



Copyright © 2008, 2026, Oracle and/or its affiliates.

Traditional Model

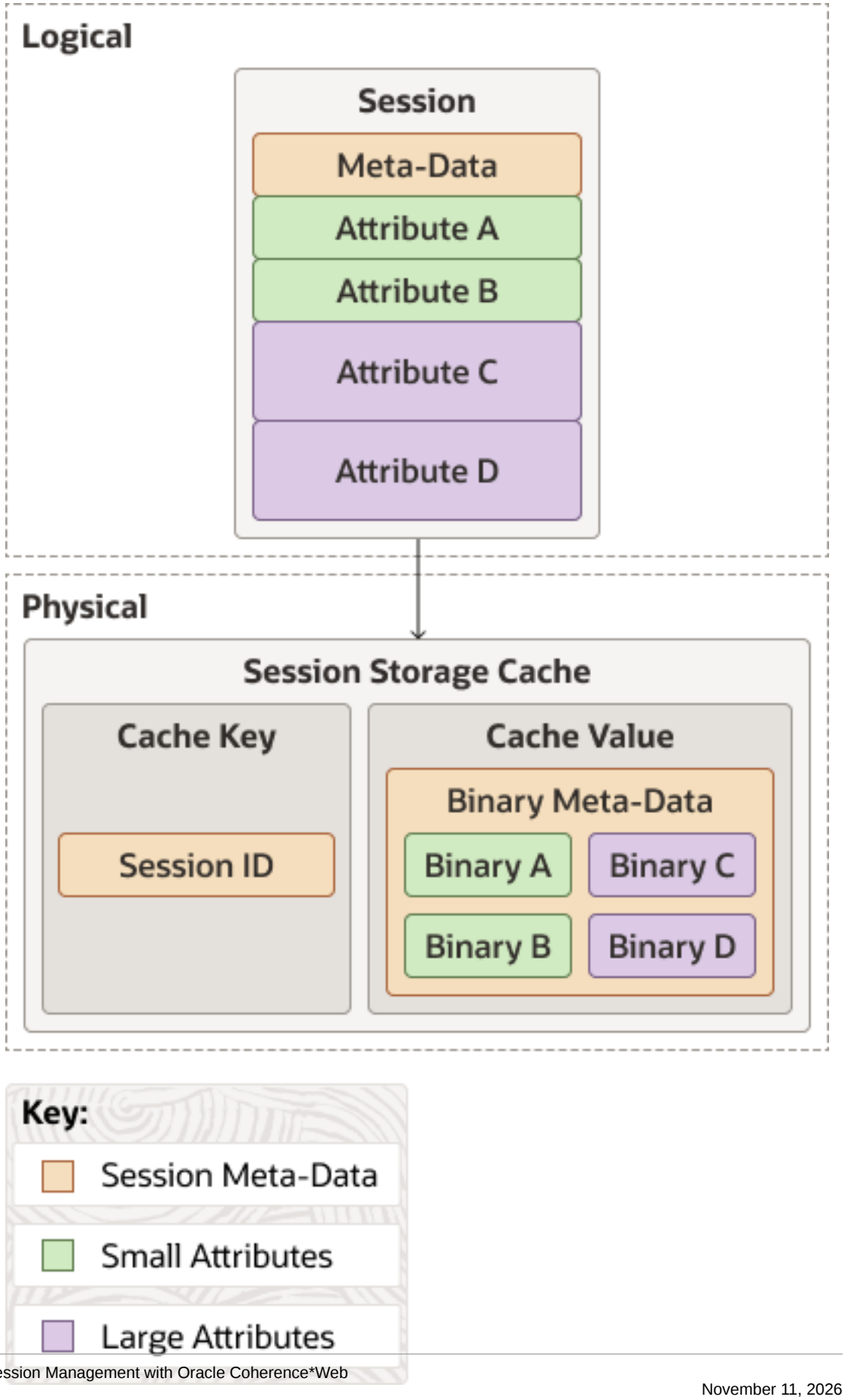
The Traditional model is represented by the `TraditionalHttpSessionModel` and `TraditionalHttpSessionCollection` objects. The `TraditionalHttpSessionCollection` object stores an HTTP session object in a single cache, but serializes each attribute independently.

This model is suggested for applications with relatively small HTTP session objects (10 KB or less) that do not have issues with object sharing between session attributes. Object sharing between session attributes occurs when multiple attributes of a session have references to the same exact object, meaning that separate serialization and deserialization of those attributes cause multiple instances of that shared object to exist when the HTTP session is later deserialized.

Figure 5-3 illustrates the relationship between the logical representation of data and its physical representation in the session storage cache. In its logical representation session data consists of metadata, and various attributes. In its physical representation in the session storage cache, the metadata and attributes are converted to binaries, and a session ID is associated with them. Note that the attributes are serialized individually instead of as a single binary BLOB (such as in the Monolithic case).

Copyright © 2008, 2026, Oracle and/or its affiliates.

Figure 5-3 Traditional Session Model



Copyright © 2008, 2026, Oracle and/or its affiliates.

Split Model

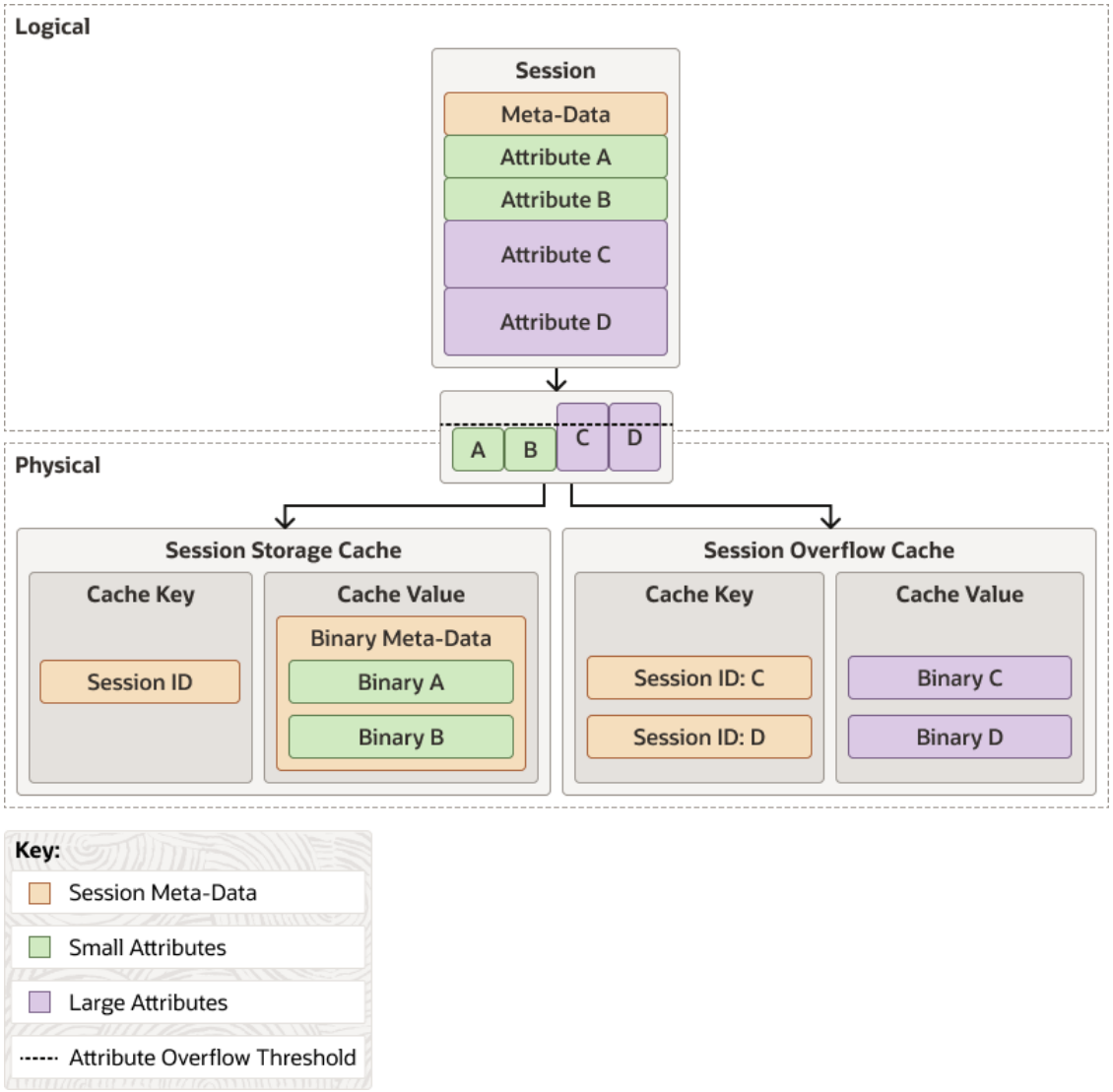
The Split model is represented by the `SplitHttpSessionModel` and `SplitHttpSessionCollection` objects. `SplitHttpSessionCollection` is the default used by Coherence*Web.

These models store the core HTTP session metadata and all of the small session attributes in the same manner as the Traditional model, thus ensuring high performance by keeping that block of binary session data small. All large attributes are split into separate cache entries to be managed individually, thus supporting very large HTTP session objects without unduly increasing the amount of data that must be accessed and updated within the cluster for each request. In other words, only the large attributes that are modified within a particular request incur any network overhead for their updates, and (because it uses near caching) the Split model generally does not incur any network overhead for accessing either the core HTTP session data or any of the session attributes.

Figure 5-4 illustrates the relationship between the logical representation of data and its physical representation in the session storage cache. In this model, large objects are stored as separate cache entries with their own session ID.

Copyright © 2008, 2026, Oracle and/or its affiliates.

Figure 5-4 Split Session Model



Session Model Recommendations

The following are recommendations on which session model to choose for your applications:

- The Split model is the recommended session model for most applications.
- The Traditional model might be more optimal for applications that are known to have small HTTP session objects.
- The Monolithic model is designed to solve a specific class of problems related to multiple session attributes that have references to the same shared object, and that must maintain that object as a shared object.

Copyright © 2008, 2026, Oracle and/or its affiliates.

> **Note**
>
> See Coherence*Web Context Parameters for descriptions of the parameters used to configure session models.

Configuring a Session Model

By default, Coherence*Web uses the split session model, where large attributes are split into separate cache entries to be managed individually. You can change the session model used by Coherence*Web by configuring the `-Dcoherence.sessioncollection.class` system property or by setting the equivalent `coherence-sessioncollection-class` context parameter in the Web application's `web.xml` file. As the value of the context parameter (or system property), use the fully-qualified class name of the `HttpSessionCollection` implementation.

- `com.tangosol.coherence.servlet.SplitHttpSessionCollection` (default) configures the Split model.
- `com.tangosol.coherence.servlet.MonolithicHttpSessionCollection` configures the Monolithic model.
- `com.tangosol.coherence.servlet.TraditionalHttpSessionCollection` configures the Traditional model.

Example 5-1 illustrates a `web.xml` entry to configure the Monolithic model.

Example 5-1 Configuring the Session Model

```
...
<context-param>
   <param-name>coherence-sessioncollection-class</param-name>
   <param-value>com.tangosol.coherence.servlet.MonolithicHttpSessionCollection</param-
value>
</context-param>
...
```

Sharing Data in a Clustered Environment

Clustering can boost scalability and availability for applications. Clustering solutions such as Coherence*Web solve many problems for developers, but successful developers must be aware of the limitations of the underlying technology, and how to manage those limitations. Understanding what the platform provides, and what users require, gives developers the ability to eliminate the gap between the two.

Session attributes must be serializable if they are to be processed across multiple JVMs, which is a requirement for clustering. It is possible to make some fields of a session attribute non-clustered by declaring those fields as transient. While this eliminates the requirement for all fields of the session attributes to be serializable, it also means that these attributes are not fully replicated to the backup server(s). Developers who follow this approach should be very careful to ensure that their applications are capable of operating in a consistent manner even if these attribute fields are lost. In most cases, this approach ends up being more difficult than simply converting all session attributes to serializable objects. However, it can be a useful pattern when very large amounts of user-specific data are cached in a session.

The Java EE Servlet specification (versions 2.2, 2.3, and 2.4) states that the servlet context should not be shared across the cluster. Non-clustered applications that rely on the servlet context as a singleton data structure have porting issues when moving to a clustered environment.

Copyright © 2008, 2026, Oracle and/or its affiliates.

A more subtle issue that arises in clustered environments is the issue of object sharing. In a non-clustered application, if two session attributes reference a common object, changes to the shared object are visible as part of both session attributes. However, this is not the case in most clustered applications. To avoid unnecessary use of compute resources, most session management implementations serialize and deserialize session attributes individually on demand. Coherence*Web (Traditional and Split session models) normally operates in this manner. If two session attributes that reference a common object are separately deserialized, the shared common object is instantiated twice. For applications that depend on shared object behavior and cannot be readily corrected, Coherence*Web provides the option of a Monolithic session model, which serializes and deserializes the entire session object as a single operation. This provides compatibility for applications that were not originally designed with clustering in mind.

Many projects require sharing session data between different Web applications. The challenge that arises is that each Web application typically has its own class loader. Consequently, objects cannot readily be shared between separate Web applications. There are two general methods used as a work around, each with its own set of trade-offs.

- Place common classes in the Java CLASSPATH, allowing multiple applications to share instances of those classes at the expense of a slightly more complicated configuration.
- Use Coherence*Web to share session data across class loader boundaries. Each Web application is treated as a separate cluster member, even if they run within the same JVM. This approach provides looser coupling between Web applications (assuming serialized classes share a common serial Version UID), but suffers from a performance impact because objects must be serialized-deserialized for transfer between cluster members.

Scalability and Performance

Moving to a clustered environment makes session size a critical consideration. Memory usage is a factor regardless of whether an application is clustered or not, but clustered applications must also consider the increased CPU and network load that larger sessions introduce. While non-clustered applications using in-memory sessions are not required to serialize-deserialize session state, clustered applications must do this every time session state is updated. Serializing session state and then transmitting it over the network becomes a critical factor in application performance. For this reason and others, a server should generally limit session size to no more than a few kilobytes.

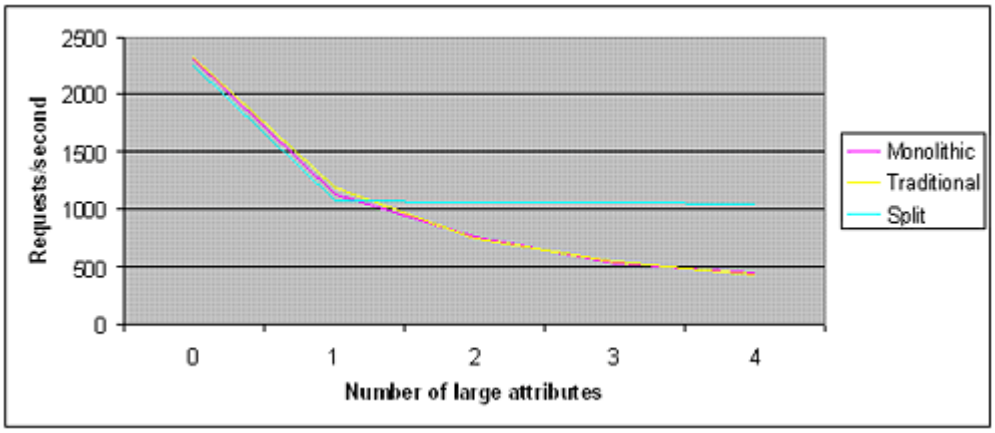
While the Traditional and Monolithic session models for Coherence*Web have the same limiting factor, the Split session model was explicitly designed to efficiently support large HTTP sessions. Using a single clustered cache entry to contain all of the small session attributes means that network traffic is minimized when accessing and updating the session or any of its smaller attributes. Independently deserializing each attribute means that CPU usage is minimized. By splitting out larger session attributes into separate clustered cache entries, Coherence*Web ensures that the application only pays the cost for those attributes when they are actually accessed or updated. Additionally, because Coherence*Web leverages the data management features of Coherence, all of the underlying features are available for managing session attributes, such as near caching, NIO buffer caching, and disk-based overflow.

Figure 5-5 illustrates performance as a function of session size. Each session consists of ten 10-character Strings and from zero to four 10,000-character Strings. Each HTTP request reads a single small attribute and a single large attribute (for cases where there are any in the session), and 50 percent of requests update those attributes. Tests were performed on a two-server cluster. Note the similar performance between the Traditional and Monolithic models; serializing-deserializing Strings consumes minimal CPU resources, so there is little performance gain from deserializing only the attributes that are actually used. The performance gain of the Split model increases to over 37:1 by the time session size reaches one megabyte

Copyright © 2008, 2026, Oracle and/or its affiliates.

(100 large Strings). In a clustered environment, it is particularly true that application requests that access only essential data have the opportunity to scale and perform better; this is part of the reason that sessions should be kept to a reasonable size.

Figure 5-5 Performance as a Function of Session Size



Another optimization is the use of transient data members in session attribute classes. Because Java serialization routines ignore transient fields, they provide a very convenient means of controlling whether session attributes are clustered or isolated to a single cluster member. These are useful in situations where data can be "lazy loaded" from other data sources (and therefore recalculated during a server failover process), and also in scenarios where absolute reliability is not critical. If an application can withstand the loss of a portion of its session state with zero (or acceptably minimal) impact on the user, then the performance benefit may be worth considering. In a similar vein, it is not uncommon for high-scale applications to treat session loss as a session timeout, requiring the user to log back in to the application (which has the implicit benefit of properly setting user expectations regarding the state of their application session).

Sticky load balancing plays a critical role because session state is not globally visible across the cluster. For high-scale clusters, user requests normally enter the application tier through a set of stateless load balancers, which redistribute (more or less randomly) these requests across a set of sticky load balancers, such as Microsoft IIS or Apache HTTP Server. These sticky load balancers are responsible for the more computationally intense act of parsing the HTTP headers to determine which server instance is processing the request (based on the server ID specified by the session cookie). If requests are misrouted for any reason, session integrity is lost. For example, some load balancers may not parse HTTP headers for requests with large amounts of POST data (for example, more than 64KB), so these requests are not routed to the appropriate server instance. Other causes of routing failure include corrupted or malformed server IDs in the session cookie. Most of these issues can be handled with proper selection of a load balancer and designing tolerance into the application whenever possible (for example, ensuring that all large POST requests avoid accessing or modifying session state).

Sticky load balancing aids the performance of Coherence*Web but is not required. Because Coherence*Web is built on the Coherence data management platform, all session data is globally visible across the cluster. A typical Coherence*Web deployment places session data in a near cache topology, which uses a partitioned cache to manage huge amounts of data in a scalable and fault-tolerant manner, combined with local caches in each application server JVM to provide instant access to commonly used session state. While a sticky load balancer is not required when Coherence*Web is used, there are two key benefits to using one. Due to the use of near cache technology, read access to session attributes is instant if user requests are consistently routed to the same server, as using the local cache avoids the cost of

Copyright © 2008, 2026, Oracle and/or its affiliates.

deserialization and network transfer of session attributes. Additionally, sticky load balancing allows Coherence to manage concurrency locally, transferring session locks only when a user request is rebalanced to another server.

Session and Session Attribute Scoping

Coherence*Web allows fine-grained control over how both session data and session attributes are scoped (or *shared*) across application boundaries.
This section includes the following topics:

- Session Scoping
- Session Attribute Scoping

Session Scoping

Coherence*Web allows session data to be shared by different Web applications deployed in the same or different Web containers. To do so, you must correctly configure the session cookie context parameters and make the classes of objects stored in session attributes available to each Web application.

If you are using cookies to store session IDs (that is, you are not using URL rewriting), you must set the session cookie path to a common context path for all Web applications that share session data. For example, to share session data between two Web applications registered under the context paths `/web/HRPortal` and `/web/InWeb`, you should set the `coherence-session-cookie-path` parameter to `/web`. On the other hand, if the two Web applications are registered under the context paths `/HRPortal` and `/InWeb`, you should set the `coherence-session-cookie-path` parameter to a slash (`/`).

If the Web applications that you would like to share session data are deployed on different Web containers running on different machines (that are not behind a common load balancer), you must also configure the session cookie domain to a domain shared by the machines. For example, to share session data between two Web applications running on `server1.example.com` and `server2.example.com`, you must set the `coherence-session-cookie-domain` context parameter to `.example.com`.

To correctly serialize or deserialize objects stored in shared sessions, the classes of all objects stored in session attributes must be available to Web applications that share session data.

> **Note**
>
> For advanced use cases where EAR cluster node-scoping or application server JVM cluster scoping is employed *and* you do not want session data shared across individual Web applications, see Preventing Web Applications from Sharing Session Data .

This section includes the following topics:

- Preventing Web Applications from Sharing Session Data
- Working with Multiple Cache Configurations
- Keeping Session Cookies Separate

Copyright © 2008, 2026, Oracle and/or its affiliates.

Preventing Web Applications from Sharing Session Data

Sometimes you might want to explicitly prevent HTTP session data from being shared by different Jakarta EE applications that participate in the same Coherence cluster. For example, assume you have two applications, `HRPortal` and `InWeb`, that share cached data in their Jakarta Enterprise Beans (EJB) tiers but use different session data. In this case, it is desirable for both applications to be part of the same Coherence cluster, but undesirable for both applications to use the same clustered service for session data. One way to do this is to use the `ApplicationScopeController` interface to define the scope of an application's attributes. Session Attribute Scoping describes this technique. Another way is to specify a unique session cache service name for each application.

Follow these steps to specify a unique session cache service name for each application:

1. Locate the `<service-name/>` elements in each `default-session-cache-config.xml` file found in your application.
2. Set the elements to a unique value for each application.

   This forces each application to use a separate clustered service for session data.
3. Include the modified `default-session-cache-config.xml` file with the application.

   Example 5-2 illustrates a sample `default-session-cache-config.xml` file for an `HRPortal` application. To prevent the `HRPortal` application from sharing session data with the `InWeb` application, rename the `<service-name>` element for the view scheme to `ViewSessionMiscHRP`. Rename the `<service-name>` element for the distributed schemes to `DistributedSessionsHRP` and `DistributedSessionsHeapOnlyHRP`.

Example 5-2 Configuration to Prevent Applications from Sharing Session Data

```
<view-scheme>
  <scheme-name>view</scheme-name>
  <back-scheme>
    <distributed-scheme>
      <scheme-ref>session-distributed-heap-only</scheme-ref>
    </distributed-scheme>
  </back-scheme>
  <reconnect-interval>30s</reconnect-interval>
  <autostart>true</autostart>
</view-scheme>

<distributed-scheme>
  <scheme-name>session-distributed</scheme-name>
  <service-name>DistributedSessions</service-name> <!-- rename this to
DistributedSessionsHRP -->
  <lease-granularity>member</lease-granularity>
  <local-storage system-property="coherence.session.localstorage">false</local-storage>
  <partition-count>257</partition-count>
  <backup-count>1</backup-count>
  <request-timeout>30s</request-timeout>
  <backing-map-scheme>
    <ramjournal-scheme>
      <high-units system-property="coherence.session.highunits"/>
      <unit-calculator>BINARY</unit-calculator>
    </ramjournal-scheme>
  </backing-map-scheme>
  <autostart>true</autostart>
</distributed-scheme>

<distributed-scheme>
```


Copyright © 2008, 2026, Oracle and/or its affiliates.

```
  <scheme-name>session-distributed-heap-only</scheme-name>
  <service-name>DistributedSessionsHeapOnly</service-name> <!-- rename this to
DistributedSessionsHeapOnlyHRP -->
  <lease-granularity>member</lease-granularity>
  <local-storage system-property="coherence.session.localstorage">false</local-storage>
  <partition-count>257</partition-count>
  <backup-count>1</backup-count>
  <request-timeout>30s</request-timeout>
  <backing-map-scheme>
    <local-scheme/>
  </backing-map-scheme>
  <autostart>true</autostart>
</distributed-scheme>
```

Working with Multiple Cache Configurations

If you are working with two or more applications running under Coherence*Web, then they could have multiple different cache configurations. In this case, the cache configuration on the cache server must contain the union of these cache configurations regardless of whether you run in storage-enabled or storage-disabled mode. This will allow the applications to be supported in the same cache cluster.

Keeping Session Cookies Separate

If you are using cookies to store session IDs, you must ensure that session cookies created by one application are not propagated to another application. To do this, you must set each application's session cookie domain and path in their `web.xml` file. To prevent cookies from being propagated, ensure that no two applications share the same context path.

For example, assume you have two Web applications registered under the context paths `/web/HRPortal` and `/web/InWeb`. To prevent the Web applications from sharing session data through cookies, set the cookie path to `/web/HRPortal` in one application, and set the cookie path to `/web/InWeb` in the other application.

If your applications are deployed on different Web containers running on separate machines, then you can configure the cookie domain to ensure that they are not in the same domain.

For example, assume you have two Web applications running on `server1.example.com` and `server2.example.com`. To prevent session cookies from being shared between them, set the cookie domain in one application to `server1.example.com`, and set the cookie domain in the other application to `server2.example.com`.

Session Attribute Scoping

In the case where sessions are shared across Web applications there are many instances where the application might scope individual session attributes so that they are either globally visible (that is, all Web applications can see and modify these attributes) or scoped to an individual Web application (that is, not visible to any instance of another application).

Coherence*Web provides the ability to control this behavior by using the `AttributeScopeController` interface. This optional interface can selectively scope attributes in cases when a session might be shared across multiple applications. This allows different applications to potentially use the same attribute names for the application-scope state without accidentally reading, updating, or removing other applications' attributes. In addition to having application-scoped information in the session, this interface allows the session to contain global (unscoped) information that can be read, updated, and removed by any of the applications that shares the session.

Copyright © 2008, 2026, Oracle and/or its affiliates.

Two implementations of the `AttributeScopeController` interface are available: `ApplicationScopeController` and `GlobalScopeController`. The `GlobalScopeController` implementation does not scope attributes, while `ApplicationScopeController` scopes all attributes to the application by prefixing the name of the application to all attribute names.

Use the `coherence-application-name` context parameter to specify the name of the application (and the Web module in which the application appears). The `ApplicationScopeController` interface will use the name of the application to scope the attributes. If you do not configure this parameter, then Coherence*Web uses the name of the class loader instead. For more information, see the description of `coherence-application-name` in Table 2-1.

> **Note**
>
> After a configured `AttributeScopeController` implementation is created, it is initialized with the name of the Web application, which it can use to qualify attribute names. Use the `coherence-application-name` context parameter to configure the name of your Web application.

This section includes the following topic:

Application requirements

Complete Preparing Session Attributes for Hardened Mode before startup and verify the intended sharing/isolation boundary. A local fallback for a nonserializable attribute does not make it available after failover; keep that availability limitation explicit in the application.

- Sharing Session Information Between Multiple Applications

Sharing Session Information Between Multiple Applications

Coherence*Web allows multiple applications to share the same session object. To do this, the session attributes must be visible to all applications. You must also specify which URLs served by WebLogic Server will be able to receive cookies.

To allow the applications to share and modify the session attributes, reference the `GlobalScopeController` (`com.tangosol.coherence.servlet.AbstractHttpSessionCollection$GlobalScopeController`) interface as the value of the `coherence-scopecontroller-class` context parameter in the `web.xml` file. `GlobalScopeController` is an implementation of the `com.tangosol.coherence.servlet.HttpSessionCollection$AttributeScopeController` interface that allows individual session attributes to be globally visible.

Example 5-3 illustrates the `GlobalScopeController` interface specified in the `web.xml` file.

Example 5-3 GlobalScopeController Specified in the web.xml File

```
<?xml version="1.0" encoding="UTF-8"?>  <web-app>    ...
    <context-param>
      <param-name>coherence-scopecontroller-class</param-name>
      <param-
value>com.tangosol.coherence.servlet.AbstractHttpSessionCollection$GlobalScopeController<
/param-value>
    </context-param>
    ...
  </web-app>
```


Copyright © 2008, 2026, Oracle and/or its affiliates.

Complete Preparing Session Attributes for Hardened Mode before startup and verify the intended sharing/isolation boundary. A local fallback for a nonserializable attribute does not make it available after failover; keep that availability limitation explicit in the application.

Cluster Node Isolation

Cluster node isolation considers the number of Coherence nodes that are created within an application server JVM and where the Coherence library is deployed.
Applications can be application server-scoped, EAR-scoped, or WAR-scoped. This section describes these considerations. For detailed information about the XML configuration for each of these options, see Configure Coherence*Web Storage Mode.

This section includes the following topics:

- Application Server-Scoped Cluster Nodes
- EAR-Scoped Cluster Nodes
- WAR-Scoped Cluster Nodes

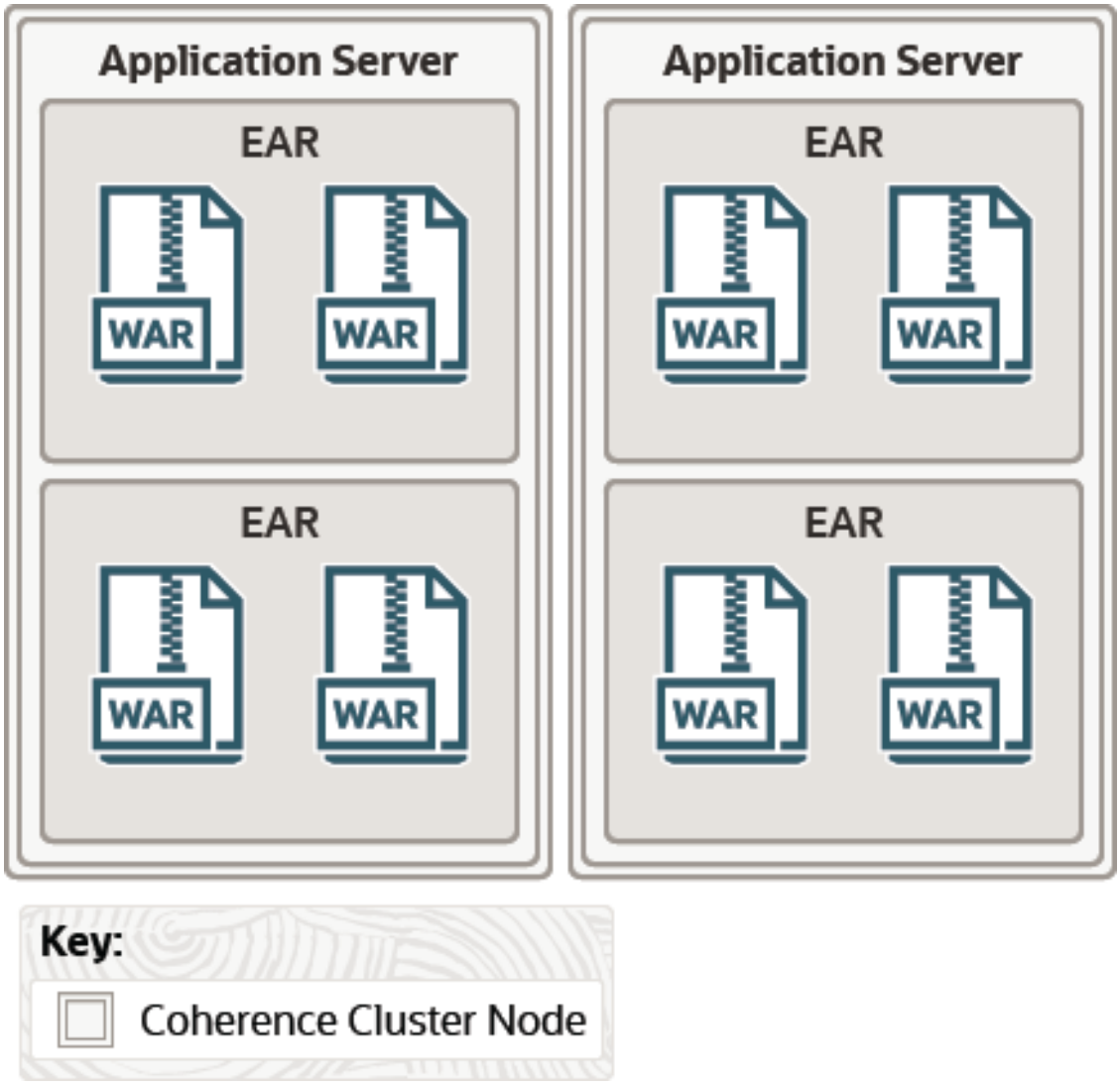
Application Server-Scoped Cluster Nodes

With this configuration, all deployed applications in a container using Coherence*Web become part of one Coherence node. This configuration produces the smallest number of Coherence nodes in the cluster (one for each Web container JVM) and, because the Coherence library (`coherence.jar`) is deployed in the container's class path, only one copy of the Coherence classes is loaded into the JVM. This minimizes the use of resources. On the other hand, because all applications are using the same cluster node, all applications are affected if one application malfunctions.

Figure 5-6 illustrates an application server-scoped cluster with two cluster nodes (application server instances). Because Coherence*Web has been deployed to each instance's class path, each instance can be considered to be a Coherence node. Each node contains two EAR files; each EAR file contains two WAR files. All of the application running in each instance share the same Coherence library and classes.

Copyright © 2008, 2026, Oracle and/or its affiliates.

Figure 5-6 Application Server-Scoped Cluster



For WebLogic Server, all Coherence*Web-enabled applications have application server scope by default. Configure Coherence*Web Storage Mode describes the XML configuration requirements for application server-scoped cluster nodes for WebLogic Server.

Copyright © 2008, 2026, Oracle and/or its affiliates.

> **Note**
>
> For platforms other than WebLogic Server, the use of application server-scoped cluster configurations should be used with care. Do not use it in environments where application interaction is unknown or unpredictable.
>
> An example of such an environment might be a deployment where multiple application teams are deploying applications written independently, without carefully coordinating and enforcing their conventions and naming standards. With this configuration, all applications are part of the same cluster—the likelihood of collisions between namespaces for caches, services, and other configuration settings is quite high and could lead to unexpected results.
>
> For these reasons, the recommended best practice is to use EAR-scoped and WAR-scoped cluster node configurations on platforms other than WebLogic Server. If you are in doubt regarding which deployment topology to choose, or if this note applies to your deployment, then *do not* choose the application server-scoped cluster node configuration.

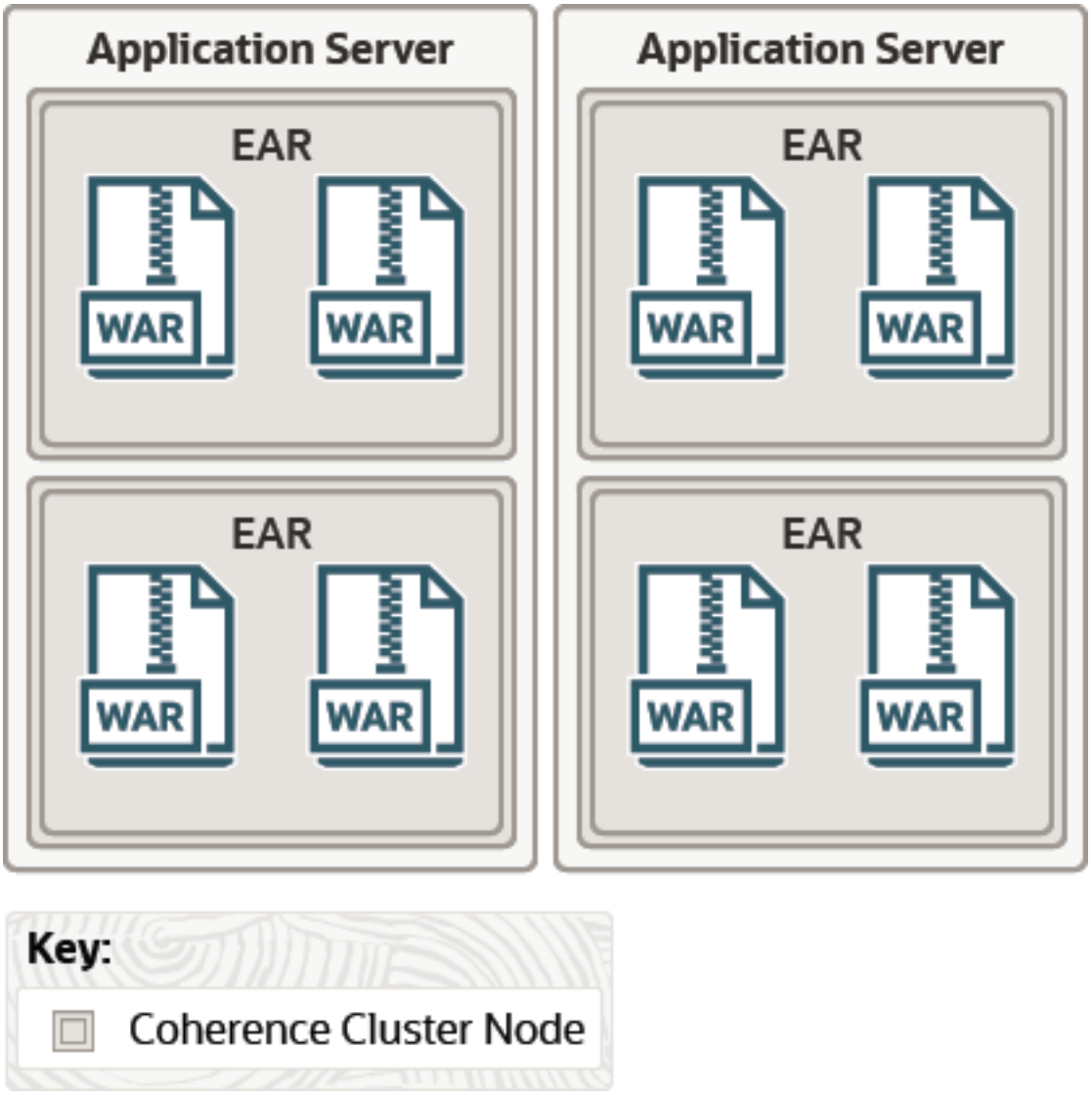
EAR-Scoped Cluster Nodes

With this configuration, all deployed applications within each EAR file become part of one Coherence node. This configuration produces one Coherence node for each deployed EAR file that uses Coherence*Web. Because the Coherence library (`coherence.jar`) is deployed in the application's classpath, only one copy of the Coherence classes is loaded for each EAR file. Since all Web applications in the EAR file use the same cluster node, all Web applications in the EAR file are affected if one of the Web applications malfunctions.

Figure 5-7 illustrates four EAR-scoped cluster nodes. Since Coherence*Web has been deployed to each EAR file, each EAR file becomes a cluster node. All applications running inside each EAR file have access to the same Coherence libraries and classes.

Copyright © 2008, 2026, Oracle and/or its affiliates.

Figure 5-7 EAR-Scoped Cluster



EAR-scoped cluster nodes reduce the deployment effort because no changes to the application server class path are required. This option is also ideal if you plan to deploy only one EAR file to an application server.

> **Note**
>
> Applications running on the WebLogic Server platform should not use EAR-scoped cluster nodes.

WAR-Scoped Cluster Nodes

With this configuration, each deployed Web application becomes its own Coherence node. This configuration produces the largest number of Coherence nodes in the cluster (one for each deployed WAR file that uses Coherence*Web) and because the Coherence library

Copyright © 2008, 2026, Oracle and/or its affiliates.

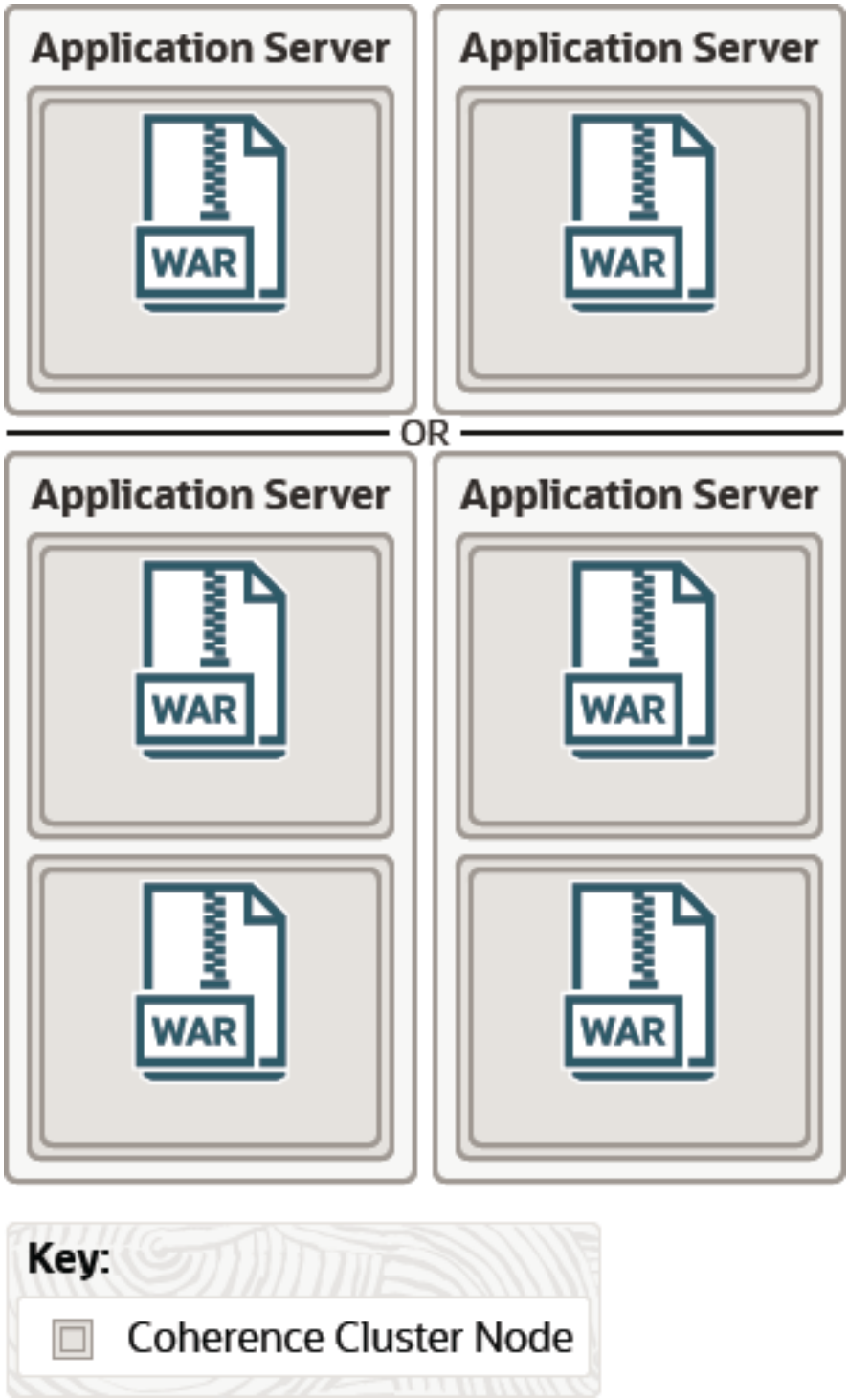
(`coherence.jar`) is deployed in the Web application's class path, there will be as many copies of the Coherence classes loaded as there are deployed WAR files. This results in the largest resource utilization of the three options. However, because each deployed Web application is its own cluster node, Web applications are completely isolated from other potentially malfunctioning Web applications.

WAR -coped cluster nodes reduce the deployment effort because no changes to the application server class path are required. This option is also ideal if you plan to deploy only one WAR file to an application server.

Figure 5-8 illustrates two different configurations of WAR files in application servers. Because each WAR file contains a copy of Coherence*Web (and Coherence), it can be considered a cluster node.

Copyright © 2008, 2026, Oracle and/or its affiliates.

Figure 5-8 WAR-Scoped Clusters



> **Note**
>
> Applications running on the WebLogic Server platform should not use WAR-scoped cluster nodes.

Copyright © 2008, 2026, Oracle and/or its affiliates.

Session Locking Modes

Oracle Coherence provides different configuration options for concurrent access to HTTP sessions.

This section includes the following topics:

- Overview of Session Locking Modes
- Optimistic Locking
- Last-Write-Wins Locking
- Member Locking
- Application Locking
- Thread Locking
- Troubleshooting Locking in HTTP Sessions
- Enabling Sticky Session Optimizations

Overview of Session Locking Modes

The following are the configuration options for concurrent access to HTTP sessions:

- Optimistic Locking, which allows concurrent access to a session by multiple threads in a single member or multiple members, while prohibiting concurrent modification.
- Last-Write-Wins Locking, which is a variation of Optimistic Locking. This allows concurrent access to a session by multiple threads in a single member or multiple members. In this case, the last write is saved. This is the default locking mode.
- Member Locking, which allows concurrent access and modification of a session by multiple threads in the same member, while prohibiting concurrent access by threads in different members.
- Application Locking , which allows concurrent access and modification of a session by multiple threads in the same Web application instance, while prohibiting concurrent access by threads in different Web application instances.
- Thread Locking, which prohibits concurrent access and modification of a session by multiple threads in a single member.

> **Note**
>
> Generally, Web applications that are part of the same cluster must use the same locking mode and sticky session optimizations setting. Inconsistent configurations could result in deadlock.

You can specify the session locking mode used by your Web applications by setting the `coherence-session-locking-mode` context parameter. Table 5-1 lists the context parameter values and the corresponding session locking modes they specify. For more information about the `coherence-session-locking-mode` context parameter, see the following sections and Coherence*Web Context Parameters.

Copyright © 2008, 2026, Oracle and/or its affiliates.

Table 5-1 Summary of coherence-session-locking-mode Context Parameter Values

Locking Mode	coherence-session-locking-mode Values
Optimistic Locking	`optimistic`
Last-Write-Wins Locking	`none`
Member Locking	`member`
Application Locking	`app`
Thread Locking	`thread`

Optimistic Locking

Optimistic Locking mode allows multiple Web container threads in one or more members to access the same session concurrently. This setting does not use explicit locking; rather an optimistic approach is used to detect and prevent concurrent updates upon completion of an HTTP request that modifies the session. The exception `ConcurrentModificationException` is thrown when the session is flushed to the cache, which is after the Servlet request has finished processing. To view the exception, set the `weblogic.debug.DebugHttpSessions` system property to `true` in the container's startup script (for example: `-Dweblogic.debug.DebugHttpSessions=true`).

The Optimistic Locking mode can be configured by setting the `coherence-session-locking-mode` parameter to `optimistic`.

Last-Write-Wins Locking

Coherence*Web is configured with Last-Write Wins Locking by default. Last-Write-Wins Locking mode is a variation on the Optimistic Locking mode. It allows multiple Web container threads in one or more members to access the same session concurrently. This setting does not use explicit locking; it does not prevent concurrent updates upon completion of an HTTP request that modifies the session. Instead, the last write, that is, the last modification made, is allowed to modify the session.

The Last-Write-Wins Locking mode can be configured by setting the `coherence-session-locking-mode` parameter to `none`. This value will allow concurrent modification to sessions with the last update being applied.

Member Locking

The Member Locking mode allows multiple Web container threads in the same cluster node to access and modify the same session concurrently, but prohibits concurrent access by threads in different members. This is accomplished by acquiring a member-level lock for an HTTP session when the session is acquired. The lock is released on completion of the of the HTTP request. See <lease-granularity> in *Developing Applications with Oracle Coherence*.

The Member Locking mode can be configured by setting the `coherence-session-locking-mode` parameter to `member`.

Application Locking

The Application Locking mode restricts session access (and modification) to threads in a single Web application instance at a time. This is accomplished by acquiring both a member-level and

Copyright © 2008, 2026, Oracle and/or its affiliates.

application-level lock for an HTTP session when the session is acquired, and releasing both locks upon completion of the HTTP request. See `<lease-granularity>` in *Developing Applications with Oracle Coherence*.

The Application Locking mode can be configured by setting the `coherence-session-locking-mode` parameter to `app`.

Thread Locking

Thread Locking mode restricts session access (and modification) to a single thread in a single member at a time. This is accomplished by acquiring both a member level, application-level, and thread-level lock for an HTTP session when the session is acquired, and releasing all three locks upon completion of the request. See `<lease-granularity>` in the distributed-scheme section of the *Developing Applications with Oracle Coherence*.

The Thread Locking mode can be configured by setting the `coherence-session-locking-mode` parameter to `thread`.

Troubleshooting Locking in HTTP Sessions

Enabling Member, Application, or Thread Locking for HTTP session access indicates that Coherence*Web will acquire a clusterwide lock for every HTTP request that requires access to a session. By default, threads that attempt to access a locked session (locked by a thread in a different member) block access until the lock can be acquired. If you want to enable a timeout for lock acquisition, configure it with the `coherence-session-get-lock-timeout` context parameter, for example:

```
...
<context-param>
    <param-name>coherence-session-get-lock-timeout</param-name>
    <param-value>30</param-value>
  </context-param>
...
```

Many Web applications do not have such a strict concurrency requirement. For these applications, using the Optimistic Locking mode has the following advantages:

- The overhead of obtaining and releasing clusterwide locks for every HTTP request is eliminated.
- Requests can be load-balanced away from failing or unresponsive members to active members without requiring the unresponsive member to release the clusterwide lock on the session.

Coherence*Web provides a diagnostic invocation service that is executed when a member cannot acquire the cluster lock for a session. You can control if this service is enabled by setting the `coherence-session-log-threads-holding-lock` context parameter. If this context parameter is set to `true` (default), then the invocation service will cause the member that has ownership of the session to log the stack trace of the threads that are currently holding the lock.

Note that the `coherence-session-log-threads-holding-lock` context parameter is available only when the `coherence-sticky-sessions` context parameter is set to `true`. This requirement exists because Coherence Web will acquire a cluster-wide lock for every session access request unless sticky session optimization is enabled. By enabling sticky session optimization, frequent lock-holding, and the subsequent production of numerous log files, can be avoided.

Copyright © 2008, 2026, Oracle and/or its affiliates.

Like all Coherence*Web messages, the Coherence `logging-config` operational configuration element controls how the message is logged. See `logging-config` in *Developing Applications with Oracle Coherence*.

Enabling Sticky Session Optimizations

If Member, Application, or Thread Locking is a requirement for a Web application that resides behind a sticky load balancer, Coherence*Web provides an optimization for obtaining the clusterwide lock required for HTTP session access. By definition, a sticky load balancer attempts to route each request for a given session to the same application server JVM that it previously routed requests to for that same session. This should be the same application server JVM that created the session. The sticky session optimization takes advantage of this behavior by retaining the clusterwide lock for a session until the session expires or until it is asked to release it. If, for whatever reason, the sticky load balancer sends a request for the same session to another application server JVM, that JVM will ask the JVM that owns the lock on the session to release the lock as soon as possible. For more information, see the `SessionOwnership` entry in Table C-2.

Sticky session optimization can be enabled by setting the `coherence-sticky-sessions` context parameter to `true`. This setting requires that Member, Application, or Thread Locking is enabled.

Deployment Topologies

Coherence*Web supports most of the same deployment topologies that Coherence does including in-process, out-of-process (that is, client/server deployment), and bridging clients and servers over Coherence*Extend.
The major supported deployment topologies are described in the following topics:

- In-Process Topology
- Out-of-Process Topology
- Out-of-Process with Coherence*Extend Topology
- Configuring Coherence*Web with Coherence*Extend

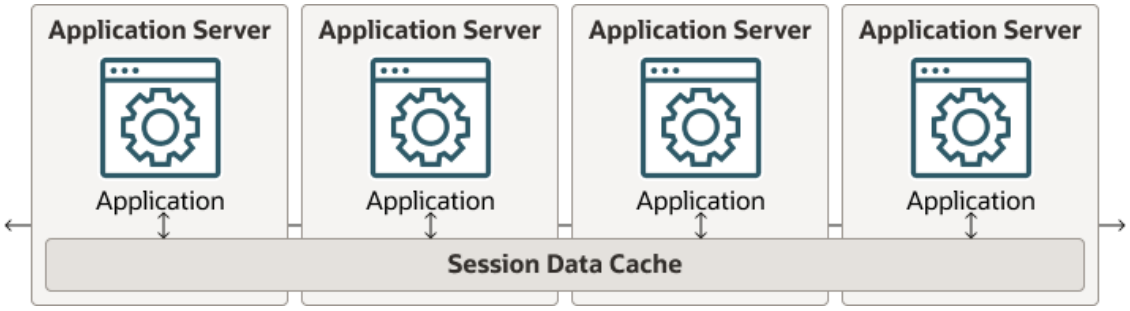
In-Process Topology

The in-process topology is not recommended for production use and is supported mainly for development and testing. By storing the session data in-process with the application server, this topology is very easy to get up and running quickly for smoke tests, developing and testing. In this topology, local storage is enabled (that is, `coherence.distributed.localstorage=true`).

Figure 5-9 illustrates the in-process topology. All of the application servers communicate with the same session data cache.

Copyright © 2008, 2026, Oracle and/or its affiliates.

Figure 5-9 In-Process Deployment Topology



Out-of-Process Topology

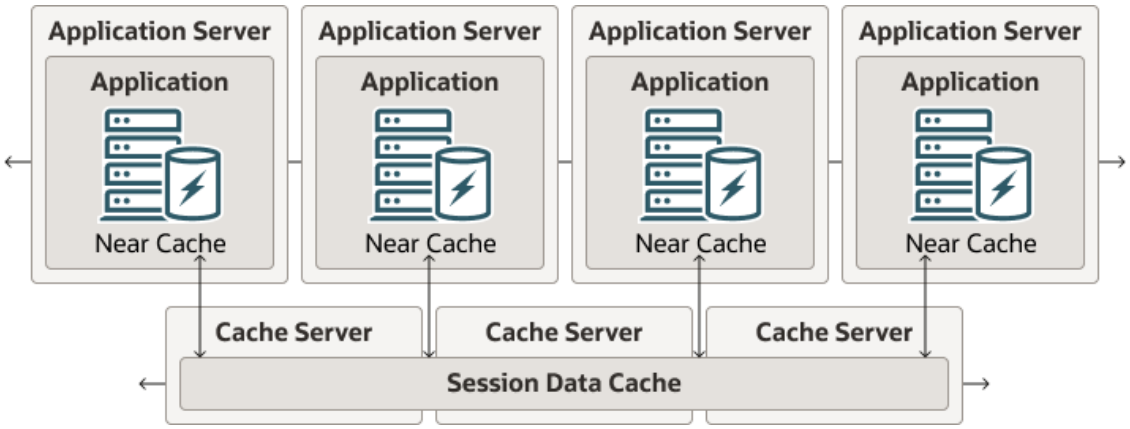
For the out-of-process deployment topology, the application servers (that is, application server tier) are configured as cache clients (that is, `coherence.distributed.localstorage=false`) and there are dedicated JVMs running as cache servers, physically storing and managing the clustered data.

This approach has these benefits:

- Session data storage is offloaded from the application server tier to the cache server tier. This reduces heap usage, garbage collection times, and so on.
- The application and cache server tiers can be scaled independently. If more application processing power is needed, just start more application servers. If more session storage capacity is needed, just start more cache servers.

The Out-of-Process topology is the default recommendation of Oracle Coherence due to its flexibility. Figure 5-10 illustrates the out-of-process topology. Each of the servers in the application tier maintain their own near cache. These near caches communicate with the session data cache which runs in a separate cache server tier.

Figure 5-10 Out-of-Process Deployment Topology



- Migrating from In-Process to Out-of-Process Topology

Copyright © 2008, 2026, Oracle and/or its affiliates.

Migrating from In-Process to Out-of-Process Topology

You can easily migrate your application from an in-process to an out of process topology. To do this, you must run a cache server in addition to the application server. Start the cache server in storage-enabled mode and ensure that it references the same session and cache configuration file (`default-session-cache-config.xml`) that the application server uses. Start the application server in storage-disabled mode. See Migrating to Out-of-Process Topology.

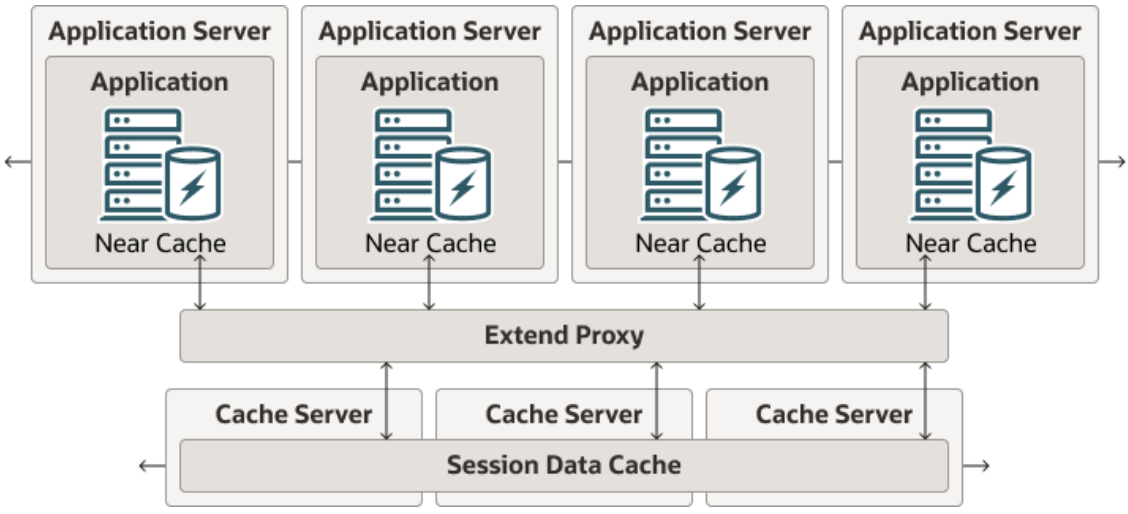
Out-of-Process with Coherence*Extend Topology

Coherence*Extend consists of two components: an extend client (or *proxy*) running outside the cluster and an extend proxy service running in the cluster hosted by one or more cache servers. The out-of-process with Coherence*Extend topology is similar to the out-of-process topology except that the communication between the application server tier and the cache server tier is over Coherence*Extend (TCP/IP). See Configuring Coherence*Web with Coherence*Extend.

This approach has the same benefits as the out-of-process topology and the ability to divide the deployment of application servers and cache servers into segments. This is ideal in an environment where application servers are on a network that does not support UDP. The cache servers can be set up in a separate dedicated network, with the application servers connecting to the cluster by using TCP.

Figure 5-11 illustrates the out-of-process with Coherence*Extend topology. Near caches in the servers in the application server tier use an extend proxy to communicate with the session data cache in the cache server tier.

Figure 5-11 Out-of-Process with Coherence*Extend Deployment Topology



Application requirements

Complete Preparing Session Attributes for Hardened Mode before startup and verify the intended sharing/isolation boundary. A local fallback for a nonserializable attribute does not make it available after failover; keep that availability limitation explicit in the application.

Copyright © 2008, 2026, Oracle and/or its affiliates.

Configuring Coherence*Web with Coherence*Extend

This section includes the following topics:

- Overview of Configuring Coherence*Web with Coherence*Extend
- Configure the Cache for Proxy and Storage JVMs
- Configure the Cache for Web Tier JVMs

Overview of Configuring Coherence*Web with Coherence*Extend

One of the deployment options for Coherence*Web is to use Coherence*Extend to connect Web container JVMs to the cluster by using TCP/IP. This configuration should be considered if any of the following situations applies:

- The Web tier JVMs are in a DMZ while the Coherence cluster is behind a firewall.
- The Web tier is in an environment that does not support UDP.
- Web tier JVMs experience long or frequent garbage collection (GC) pauses.
- Web tier JVMs are restarted frequently.

In these deployments, there are three types of participants:

- Web tier JVMs, which are the Extend clients in this topology. They are not members of the cluster; instead, they connect to a proxy node in the cluster that will issue requests to the cluster on their behalf.
- Proxy JVMs, which are storage-disabled members (nodes) of the cluster that accept and manage TCP/IP connections from Extend clients. Requests that arrive from clients will be sent into the cluster, and responses will be sent back through the TCP/IP connections.
- Storage JVMs, which are used to store the actual session data in memory.

To Configure Coherence*Web to Use Coherence*Extend

1. Configure Coherence*Web to use the Optimistic Locking mode. See Optimistic Locking.
2. Configure a cache configuration file for the proxy and storage JVM. See Configure the Cache for Proxy and Storage JVMs.
3. Modify the Web tier cache configuration file to point to one or more of the proxy JVMs. See Configure the Cache for Web Tier JVMs.

Configure the Cache for Proxy and Storage JVMs

The session cache configuration file (`WEB-INF/classes/default-session-cache-config.xml`) is an example Coherence*Web session cache configuration file that uses Coherence*Extend. Use this file for the proxy and storage JVMs.

To configure a cache Proxy and storage JVMs, use the `<proxy-scheme>` element and use a `<tcp-acceptor>` element to define the host and port to which the web tier extend client connects. The following example configures a proxy to listen on the local host at port 9099.

```
<proxy-scheme>
   <service-name>SessionProxy</service-name>
   <acceptor-config>

      <tcp-acceptor>
```


Copyright © 2008, 2026, Oracle and/or its affiliates.

```
         <local-address>
            <address system-
property="coherence.session.proxy.localhost">localhost</address>
            <port system-property="coherence.session.proxy.localport">9099</
port>
         </local-address>
      </tcp-acceptor>
<serializer>
         <instance>
            <class-name>com.tangosol.io.DefaultSerializer</class-name>
         </instance>
      </serializer>
   </acceptor-config>
   <autostart system-property="coherence.session.proxy">true</autostart>
</proxy-scheme>
```

The above example defines system property overrides that allow the same file to be used for both proxy and storage JVMs.

When used by a proxy JVM, the system properties described in Table 5-2 should be specified.

> **Note**
>
> If you are writing applications for the WebLogic Server platform and you are using a customized session cache configuration file, then the file must be packaged in a GAR file for deployment. See Using a Custom Session Cache Configuration File.
>
> For more information on the packaging requirements for a GAR file, see also Packaging Coherence Applications for WebLogic Server in *Administering Oracle Coherence* and Creating Coherence Applications for Oracle WebLogic Server in *Developing Oracle Coherence Applications for Oracle WebLogic Server*.

Table 5-2 System Property Values for Proxy JVMs

System Property Name	Value
`coherence.session.localstorage`	`false`
`coherence.session.proxy`	`true`
`coherence.session.proxy.localhost`	The host name or IP address of the NIC to which the proxy will bind.
`coherence.session.proxy.localport`	A unique port number to which the proxy will bind.

When used by a cache server, specify the system properties described in Table 5-3.

Table 5-3 System Property Values for Storage JVMs

System Property Name	Value
`coherence.session.localstorage`	`true`
`coherence.session.proxy`	`false`

Copyright © 2008, 2026, Oracle and/or its affiliates.

Configure the Cache for Web Tier JVMs

Coherence*Extend clients must also include a session cache configuration file. The file can be based on the `default-session-cache-config.xml` file that is found in the `coherence-web.jar` file.

To Install the Session Cache Configuration File for the Web Tier:

1. Extract the `default-session-cache-config.xml` file from the `coherence-web.jar` file.
2. Modify the `session-storage` cache mapping definition and rename the `<scheme-name>` value from `session-distributed` to `session-remote`.

```
<cache-mapping>
   <cache-name>session-storage</cache-name>
   <scheme-name>session-remote</scheme-name>
</cache-mapping>
```

3. Define the `session-remote` caching scheme using the `<remote-cache-scheme>` element and add a proxy address (host and port) within the `<remote-addresses>` element. In most cases, you should include the IP address and port of all proxy JVMs to ensure load balancing and failover. The following example configures the web tier to connect to a proxy that is running the on the local host at port 9099. Change the address as required for your proxy.

```
<remote-cache-scheme>
   <scheme-name>session-remote</scheme-name>
   <initiator-config>

      <tcp-initiator>
         <remote-addresses>
            <socket-address>
               <address>localhost</address>
               <port>9099</port>
            </socket-address>
         </remote-addresses>
      </tcp-initiator>
<serializer>
         <instance>
            <class-name>com.tangosol.io.DefaultSerializer</class-name>
         </instance>
      </serializer>
   </initiator-config>
</remote-cache-scheme>
```

> **Note**
>
> The `<remote-addresses>` element contains the proxy server(s) to which the Web container connects. By default, the Web container will pick an address at random (if there is more than one address in the configuration). If the connection between the Web container and the proxy is broken, the container connects to another proxy in the list.

Copyright © 2008, 2026, Oracle and/or its affiliates.

4. Rename the file to `default-session-cache-config-web-tier.xml`.
5. Place the file in the `WEB-INF/classes` directory of your Web application. If you used the WebInstaller to integrate Coherence*Web, replace the existing file that was added by the WebInstaller.

Accessing Sessions with Lazy Acquisition

Lazy acquisition can be enabled to avoid unnecessary acquiring of a session whenever a servlet or filter is called.

By default, Web applications instrumented with the WebInstaller will always acquire a session whenever a servlet or filter is called. The session is acquired regardless of whether the servlet or filter actually needs a session. This can be expensive in terms of time and processing power if you run many servlets or filters that do not require a session.

To avoid this behavior, enable lazy acquisition by setting the `coherence-session-lazy-access` context parameter to `true` in the `web.xml` file. The session will be acquired only when the servlet or filter attempts to access it.

Overriding the Distribution of HTTP Sessions and Attributes

The `HttpSessionCollection.SessionDistributionController` interface can be used to override the default distribution of HTTP sessions and attributes in a Web application. This section includes the following topics:

- Overview of Overriding HTTP Session Distribution
- Implementing a Session Distribution Controller
- Registering a Session Distribution Controller Implementation

Overview of Overriding HTTP Session Distribution

You override the default distribution by setting the `coherence-distributioncontroller-class` context parameter (see Registering a Session Distribution Controller Implementation). The value of the context parameter indicates an implementation of the `SessionDistributionController` interface.

An implementation of the `SessionDistributionController` interface can identify sessions or attributes in any of the following ways:

- Distributed, where a *distributed* session or attribute is stored within the Coherence data grid, and thus, accessible to other server JVMs. All sessions (and their attributes) are managed in a distributed manner. This is the default behavior and is provided by the `com.tangosol.coherence.servlet.AbstractHttpSessionCollection$DistributedController` implementation of the `SessionDistributionController` interface.
- Local, where a *local* session or attribute is stored on the originating server's heap, and thus, only accessible by that server. The `com.tangosol.coherence.servlet.AbstractHttpSessionCollection$LocalController` class provides this behavior. This option is not recommended for production purposes, but it can be useful for testing the difference in scalable performance between local-only and fully distributed implementations.
- Hybrid, which is similar to distributed in that all sessions and serializable attributes are managed in a distributed manner. However, unlike distributed, session attributes that do not implement the `Serializable` interface will be kept local. The

Copyright © 2008, 2026, Oracle and/or its affiliates.

`com.tangosol.coherence.servlet.AbstractHttpSessionCollection$HybridController` class provides this behavior.

At any point during the life of a session, the session or attributes for that session can change from local or distributed. However, when a session or attribute is distributed it cannot change back to local.

You can use the Session Distribution Controller in any of the following ways:

- You can allow new sessions to remain local until you add an attribute (for example, when you add the first item to an online shopping cart); the idea is that a session must be fault-tolerant only when it contains valuable data.
- Some Web frameworks use session attributes to store the UI rendering state. Often, this data cannot be distributed because it is not serializable. Using the Session Distribution Controller, these attributes can be kept local while allowing the rest of the session attributes to be distributed.
- The Session Distribution Controller can assist in the conversion from nondistributed to distributed systems, especially when the cost of distributing all sessions and all attributes is a consideration.

Implementing a Session Distribution Controller

Example 5-4 illustrates a sample implementation of the `HttpSessionCollection.SessionDistributionController` interface. In the sample, sessions are tested to see if they have a shopping cart attached (only these sessions will be distributed). Next, the session is tested whether it contains a certain attribute. If the attribute is found, then it is not distributed.

Example 5-4 Sample Session Distribution Controller Implementation

```
import com.tangosol.coherence.servlet.HttpSessionCollection;
import com.tangosol.coherence.servlet.HttpSessionModel;

/**
* Sample implementation of SessionDistributionController
*/
public class CustomSessionDistributionController
        implements HttpSessionCollection.SessionDistributionController
    {
    public void init(HttpSessionCollection collection)
        {
        }

    /**
    * Only distribute sessions that have a shopping cart.
    *
    * @param model Coherence representation of the HTTP session
    *
    * @return true if the session should be distributed
    */
    public boolean isSessionDistributed(HttpSessionModel model)
        {
        return model.getAttribute("shopping-cart") != null;
        }

    /**
    * If a session is "distributed", then distribute all attributes with the
    * exception of the "ui-rendering" attribute.
    *
```


Copyright © 2008, 2026, Oracle and/or its affiliates.

```
* @param model Coherence representation of the HTTP session
* @param sName name of the attribute to check
*
* @return true if the attribute should be distributed
*/
public boolean isSessionAttributeDistributed(HttpSessionModel model,
        String sName)
    {
    return !"ui-rendering".equals(sName);
    }
}
```

Registering a Session Distribution Controller Implementation

After you have written your `SessionDistributionController` implementation, you can register it with your application by using the `coherence-distributioncontroller-class` context parameter. See Coherence*Web Context Parameters.

Detecting Changed Attribute Values

Coherence*Web tracks if attributes retrieved from the session have changed during the course of processing a request. This is done by caching the initial serialized binary form of the attribute when it is retrieved from the session. At the end of processing a request, Coherence*Web compares the current binary value of the attribute with the old version of the binary. If the values do not match, then the current value is written to the cache. If your application does not mutate session attributes without doing a corresponding `set`, then you should set the `coherence-enable-suspect-attributes` context parameter to `false`. This improves memory use and optimizes near-caching.

Saving Non-Serializable Attributes Locally

Coherence*Web attempts to serialize all session attributes using `java.io.Serializable`. Coherence*Web does not support the use of Coherence's Portable Object Format (POF) to serialize session attributes. If you are working with session attributes that are not serializable, then store them locally by setting the `coherence-preserve-attributes` parameter to `true`. This parameter requires a load balancer to retrieve non-serializable attributes for a session. Note that an application must be able to recover if the client (application server) fails and the attributes are lost. See Coherence*Web Context Parameters.
Complete Preparing Session Attributes for Hardened Mode before startup and verify the intended sharing/isolation boundary. A local fallback for a nonserializable attribute does not make it available after failover; keep that availability limitation explicit in the application.

Securing Coherence*Web Deployments

Coherence provides a Transport Layer Security (TLS) implementation to prevent unauthorized Coherence TCMP cluster members from accessing HTTP session cache servers.
This implementation can be used to secure TCMP communication between cluster nodes and TCP communication between Coherence*Extend clients and proxies.

Copyright © 2008, 2026, Oracle and/or its affiliates.

> **Note**
>
> Although the terms TLS, SSL, and SSL/TLS are used interchangeably throughout Coherence documentation, it is expected and encouraged that you use a currently supported version of TLS, not *SSL*, to secure Coherence*Web deployments.

This section provides only an overview of using SSL in a Coherence environment. See Using TLS to Secure Communication in *Securing Oracle Coherence*.

Using SSL to Secure TCMP Communications

A Coherence cluster can be configured to use SSL with TCMP. Coherence allows you to use both one-way and two-way authentication. Two-Way authentication is typically used more often than one-way authentication, which has fewer use cases in a cluster environment. In addition, it is important to realize that TCMP is a peer-to-peer protocol that generally runs in trusted environments where many cluster nodes are expected to remain connected with each other. The implications of SSL on administration and performance should be carefully considered.

In this configuration, you can use the pre-defined, out-of-the-box SSL socket provider that allows for two-way communication SSL connections based on peer trust, or you can define your own SSL socket provider.

Using SSL to Secure Extend Client Communication

Communication between extend clients and extend proxies can be secured using SSL. SSL requires configuration on both the client side as well as the cluster side. On the cluster side, you configure SSL in the cluster-side cache configuration file by defining a SSL socket provider for a proxy service. You can define the SSL socket provider either for all proxy services or for individual proxy services.

On the client side, you configure SSL in the client-side cache configuration file by defining a SSL socket provider for a remote cache scheme and, if required, for a remote invocation scheme. Like the cluster side, you can define the SSL socket provider either for all remote services or for individual remote services.

Application requirements

Complete Preparing Session Attributes for Hardened Mode before startup and verify the intended sharing/isolation boundary. A local fallback for a nonserializable attribute does not make it available after failover; keep that availability limitation explicit in the application.

Customizing the Name of the Session Cache Configuration File

Coherence*Web uses the `default-session-cache-config.xml` file to configure session caches. You can specify a different file using the `coherence-cache-configuration-path` context parameter in the `web.xml` file.
The following example configures Coherence*Web to use the `my-default-session-cache-config-name.xml` file for session cache configuration:

```
...
<context-param>
   <param-name>coherence-cache-configuration-path</param-name>
   <param-value>my-default-session-cache-config-name.xml</param-value>
</context-param>
...
```


Copyright © 2008, 2026, Oracle and/or its affiliates.

Configuring Logging for Coherence*Web

Coherence*Web uses the logging framework provided by Coherence. Coherence has its own logging framework and also supports the use of Log4j2, SLF4J, and Java logging to provide a common logging environment for an application.
Logging in Coherence occurs on a dedicated and low-priority thread to reduce the impact of logging on the critical portions of the system. Logging is pre-configured and the default settings should be changed as required. See Configuring Logging in *Developing Applications with Oracle Coherence*.

The Coherence*Web logging level can also be set using the context parameter/system property `coherence-session-logger-level`. This is an alternative way to set the logging level for Coherence*Web (as opposed to using JDK logging). See Coherence*Web Context Parameters for more information on this parameter.

> **Warning**
>
> Applications that use the JDK logging framework can configure Coherence to use JDK logging as well. Note, however, that setting the log level to FINEST can expose session IDs in the log file.

Getting Concurrent Access to the Same Session Instance

A cache delegator is used for applications that require concurrent access to the same session instance.

A cache delegator class is a class that is responsible for manipulating (getting, putting, or deleting) any data in the distributed cache. Use the `<coherence-cache-delegator-class>` context parameter in the `web.xml` file to specify the name of the class responsible for the data manipulation.

One of the possible values for the context parameter is the `com.tangosol.coherence.servlet.LocalSessionCacheDelegator` class. This class indicates that the local cache should be used for storing and retrieving the session instance before attempting to use the distributed cache. This delegator is useful for applications that require concurrent access to the same session instance.

> **Note**
>
> This feature must be enabled when working with PeopleSoft applications.

To enable the `LocalSessionCacheDelegator` cache delegator, the following items must be configured in `web.xml`:

- The `coherence-cache-delegator-class` context parameter with the value set to `com.tangosol.coherence.servlet.LocalSessionCacheDelegator`.
- The `coherence-preserve-attributes` context parameter set to `true` to allow nonserializable objects to be stored in the session object.

Copyright © 2008, 2026, Oracle and/or its affiliates.

- The `coherence-distributioncontroller-class` context parameter with the value set to `com.tangosol.coherence.servlet.AbstractHttpSessionCollection$HybridController`. This value forces all sessions and serializable attributes to be managed in a distributed manner. All session attributes that do not implement the `Serializable` interface will be kept local. Note that the use of this context parameter also requires `coherence-sticky-sessions` optimization to be enabled.

Example 5-5 illustrates a sample configuration for the cache delegator in the `web.xml` file.

Example 5-5 Configuring Cache Delegator in the web.xml File

```
...
 <context-param>
    <param-name>coherence-cache-delegator-class</param-name>
    <param-value>com.tangosol.coherence.servlet.LocalSessionCacheDelegator
</param-value>
 </context-param>
 <context-param>
    <param-name>coherence-preserve-attributes</param-name>
    <param-value>true</param-value>
 </context-param>
 <context-param>
    <param-name>coherence-distributioncontroller-class</param-name>
    <param-
value>com.tangosol.coherence.servlet.AbstractHttpSessionCollection$HybridController</
param-value>
 </context-param>
...
```

Federated Session Caches

The Coherence federated caching feature replicates cache data asynchronously across multiple geographically dispersed clusters.
Coherence*Web can take advantage of federated caching to provide redundancy, off-site backup, and multiple points of access for application users that are in different geographical locations. For details about Federated caching, see *Administering Oracle Coherence*.

To use federated caching for HTTP session caches:

1. Define federation cluster participants and a federation topology. See Configuring Cache Federation.
2. Enable federated caching for HTTP session management. See Enabling the Coherence Session Cache in WebLogic Remote Console .
3. Configure Coherence*Web to use a federated cache scheme. A default session cache configuration file is included in the `coherence-web.jar` library and is called `default-federated-session-cache-config.xml` session cache configuration file. To use the default federated session cache configuration file, use the `coherence-session-cache-federated` context parameter with the value set to `true`.
4. (Optional) The default federated topology that is configured is automatically used if no topology is configured, to explicitly specify a topology, update or override the `default-federated-session-cache-config.xml` session cache configuration file and associate the default federated caching scheme (`session-distributed`) with a federation topology. See Associating a Federated Cache with a Federation Topology in *Administering Oracle Coherence*. For example:

   ```
   <federated-scheme>
      <scheme-name>session-distributed</scheme-name>
      <service-name>FederatedDistributedSessions</service-name>
   ```


Copyright © 2008, 2026, Oracle and/or its affiliates.

```
      <thread-count system-property="coherence.session.threads">4
         </thread-count>
      <lease-granularity>member</lease-granularity>
      <local-storage system-property="coherence.session.localstorage">
         false</local-storage>
      <partition-count>257</partition-count>
      <backup-count>1</backup-count>
      <request-timeout>30s</request-timeout>
      <backing-map-scheme>
         <ramjournal-scheme>
            <high-units system-property="coherence.session.highunits"/>
            <unit-calculator>BINARY</unit-calculator>
         </ramjournal-scheme>
      </backing-map-scheme>
      <autostart>true</autostart>
      <topologies>
         <topology>
            <name>MyTopology</name>
            <cache-name>fed-remote</cache-name>
         </topology>
      </topologies>
   </federated-scheme>
```

Preparing Session Attributes for Hardened Mode

Package compatible session models and generated trust metadata on every receiving session tier.

A session attribute crosses an application trust boundary when it is serialized, replicated, federated or recovered on another process. Java Serializable or a POF type ID alone does not admit that class in hardened mode. Coherence 26.1 defaults to hardened when coherence.security.mode is omitted.

- Inventory every session key/value/model and nested object graph, including framework session state, listeners and custom serialization. Avoid storing live connections, container objects, credentials or arbitrary executable definitions in sessions.
- Generate application security metadata during the build and package META-INF/coherence/security-config.xml with the classes visible to the session runtime. Deploy compatible model/POF definitions and metadata to every deserializing application/proxy/storage process.
- Keep class/serializer evolution compatible during rolling redeployment and failover. Test a session created by each participating application version on the other versions.
- Use explicit application/scoping/cache configuration to avoid accidental sharing. A shared cluster, service, scope or cookie path does not establish authorization between applications or tenants.
- Protect cluster/Extend transport and authenticate intended peers. Secure browser-to-container HTTPS and cookie handling separately; backend TLS does not secure the browser cookie.
- Before opening traffic, test a valid session and rejection of an unregistered type or disallowed executable. Diagnose policy and loader visibility instead of disabling hardened mode.

Follow Building Examples with Application Security Metadata and Securing Serialization and Remote Execution for the complete build/type/executable procedures. Protect persisted or archived session data as described in Limiting Resource Use and Protecting Persisted Data.

Copyright © 2008, 2026, Oracle and/or its affiliates.

6

Monitoring Applications

You can use JMX MBeans to monitor the health and performance of Coherence*Web on your system. Coherence*Web management includes support for the Coherence *Reporter*—a JMX-based reporting utility that provides several preconfigured reports that help administrators and developers manage capacity and troubleshoot problems.

> **Note**
>
> To enable Coherence*Web JMX Management and Monitoring, this section assumes that you have first set up the Coherence Clustered JMX Framework. See Using JMX to Manage Coherence in *Managing Oracle Coherence*.

This chapter includes the following sections:

Correlate session/reaper/cache metrics with the underlying service health and backlog. Protect JMX/HTTP observation endpoints and retain application/service/member dimensions; a live web container does not prove that distributed sessions are ready or safe. See Using Oracle Coherence Metrics and Verifying Session Expiry Failover and Cleanup.

- Managing and Monitoring Applications with JMX
- Managing and Monitoring Applications on WebLogic Server
- Running Performance Reports

Managing and Monitoring Applications with JMX

The management attributes and operations for Web applications that use Coherence*Web are visible through the `HttpSessionManagerMBean` MBean. During startup, each Coherence*Web Web application registers a single instance of the `HttpSessionManager` class. You can use a monitoring tool, such as JConsole, to view the values of the MBean attributes. The MBean is unregistered when the Web application shuts down.
Table 6-1 describes the object name that the MBean uses for registration.

Table 6-1 Object Name for HttpSessionManagerMBean

Managed Bean	Object Name
`HttpSessionManager`	`type=HttpSessionManager`, `nodeId=cluster node id`, `appId=web application id`

Table 6-2 describes the information that `HttpSessionManager` provides. All of the names represent attributes, except `resetStatistics`, which is an operation.

Several of the MBean attributes use the following prefixes:

Copyright © 2008, 2026, Oracle and/or its affiliates.

- `LocalSession`, which indicates a session that is not distributed to all members of the cluster. The session remains *local* to the originating server until a later point in the life of the session.
- `LocalAttribute`, which indicates a session attribute that is not distributed to all members of the cluster.
- `Overflow`, a cache which stores the large session attributes when the Split Session model is used.

Table 6-2 Information Returned by the HttpSessionManager

Attribute Name	Data Type	Description
`AverageReapDuration`	long	The average reap duration (the time it takes to complete a reap cycle) in milliseconds, since the statistic was reset. See Getting Session Reaper Performance Statistics.
`CollectionClassName`	String	The fully qualified class name of the `HttpSessionCollection` implementation in use. The `HttpSessionCollection` interface is an abstract model for a collection of `HttpSessionModel` objects. The interface is not at all affected by how the sessions communicate between the clients and the servers.
`FactoryClassName`	String	The fully-qualified class name of the `Factory` implementation being used. The `SessionHelper.Factory` class is used by the `SessionHelper` class to obtain objects that implement various important parts of the servlet specification. The `Factory` implementation can be placed in front of the application instead of the application server's own objects. This changes the apparent implementation of the application server itself (for example, adding clustering.)
`LastReapDuration`	long	The amount of time, in milliseconds, it took for the last reap cycle to finish. See Getting Session Reaper Performance Statistics.
`LocalAttributeCacheName`	String	The name of the local cache that stores non-distributed session attributes. If the attribute displays null then local session attribute storage is disabled.
`LocalAttributeCount`	Integer	The number of non-distributed session attributes stored in the local session attribute cache. If the attribute displays `-1`, then local session attribute storage is disabled.
`LocalSessionCacheName`	String	The name of the local cache that stores nondistributed sessions. If the attribute displays a null value, then local session storage is disabled.
`LocalSessionCount`	Integer	The number of nondistributed sessions stored in the local session cache. If the attribute displays a -1 value, then local session storage is disabled.
`MaxReapedSessions`	long	The maximum number of sessions reaped in a reap cycle since the statistic was reset. See Getting Session Reaper Performance Statistics.
`NextReapCycle`	java.lang.Date	The time, expressed as a `java.lang.Date` data type, for the next reap cycle. See Getting Session Reaper Performance Statistics.
`OverflowAverageSize`	Integer	The average size (in bytes) of the session attributes stored in the overflow clustered cache since the last time statistics were reset. If the attribute displays `-1`, then a `SplitHttpSessionCollection` model is not in use.

Copyright © 2008, 2026, Oracle and/or its affiliates.

Table 6-2 (Cont.) Information Returned by the HttpSessionManager

Attribute Name	Data Type	Description
`OverflowCacheName`	String	The name of the clustered cache that stores the large attributes that exceed a certain size and thus are determined to be more efficiently managed as separate cache entries and not as part of the serialized session object itself. A null value is displayed if a `SplitHttpSessionCollection` model is not in use.
`OverflowMaxSize`	Integer	The maximum size (in bytes) of a session attribute stored in the overflow clustered cache since the last time statistics were reset. The attribute displays a -1 value if a `SplitHttpSessionCollection` model is not in use.
`OverflowThreshold`	Integer	The minimum length (in bytes) that the serialized form of an attribute value must be stored in the separate overflow cache that is reserved for large attributes. The attribute displays a -1 value if a `SplitHttpSessionCollection` model is not in use.
`OverflowUpdates`	Integer	The number of updates to session attributes stored in the overflow clustered cache since the last time statistics were reset. The attribute displays a -1 value if a `SplitHttpSessionCollection` model is not in use.
`ReapedSessions`	long	The number of sessions reaped during the last cycle. See Getting Session Reaper Performance Statistics.
`ReapedSessionsTotal`	long	The number of expired sessions that have been reaped since the statistic was reset. See Getting Session Reaper Performance Statistics.
`ServletContextCacheName`	String	The name of the clustered cache that stores `jakarta.servlet.ServletContext` attributes. The attribute displays `null` if `ServletContext` is not clustered.
`ServletContextName`	String	The name of the Web application `ServletContext`.
`SessionAverageLifetime`	Integer	The average lifetime (in seconds) of session objects invalidated (either due to expiration or to an explicit invalidation) since the last time statistics were reset.
`SessionAverageSize`	Integer	The average size (in bytes) of session objects placed in the session storage clustered cache since the last time statistics were reset.
`SessionCacheName`	String	The name of the clustered cache that stores serialized session objects.
`SessionDebugLogging`	Boolean	A flag to indicate whether the Coherence*Web session debug logging is enabled (`true`) or not (`false`). Set the attribute to `true` to enabled session debug logging or `false` to disable it.
`SessionIdLength`	Integer	The length (in characters) of generated session IDs.
`SessionMaxSize`	Integer	The maximum size (in bytes) of a session object placed in the session storage clustered cache since the last time statistics were reset.
`SessionMinSize`	Integer	The minimum size (in bytes) of a session object placed in the session storage clustered cache since the last time statistics were reset.
`SessionStickyCount`	Integer	The number of session objects that belong to this instance of the Web application. The attribute displays `-1` if sticky session optimizations are disabled.
`SessionTimeout`	Integer	The session expiration time (in seconds). The attribute displays `-1` if sessions never expire.
`SessionUpdates`	Integer	The number of updates of session object stored in the session storage clustered cache since the last time statistics were reset.

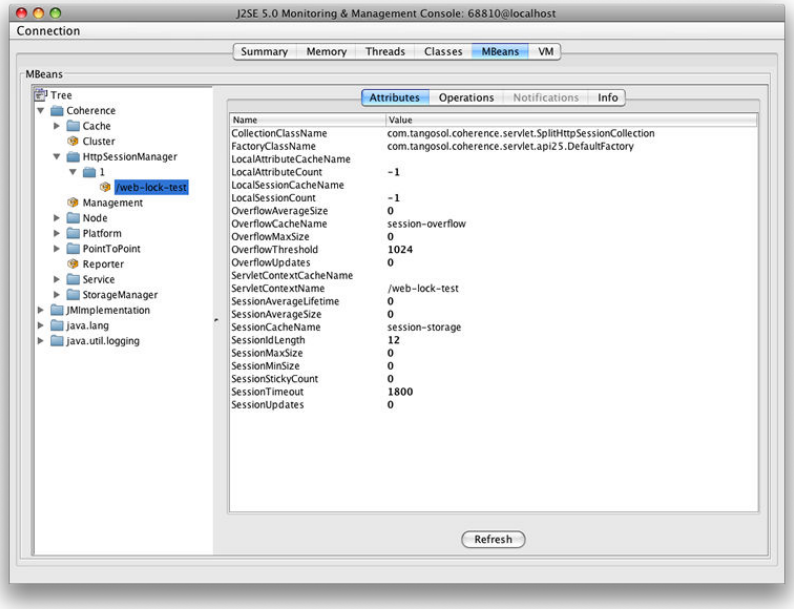
Copyright © 2008, 2026, Oracle and/or its affiliates.

Table 6-2 (Cont.) Information Returned by the HttpSessionManager

Attribute Name	Data Type	Description
`resetStatistics` (operation)	void	Reset the session management statistics.

Figure 6-1 illustrates the attributes of the `HttpSessionManager` MBean displayed in the JConsole monitoring tool.

Figure 6-1 HttpSessionManager Displayed in the JConsole Monitoring Tool



Managing and Monitoring Applications on WebLogic Server

For WebLogic Server, management attributes and operations for Web applications that use Coherence*Web are visible through the `WebLogicHttpSessionManagerMBean` MBean. Table 6-3 describes the object name that the MBean uses for registration.

Table 6-3 Object Name for WebLogicHttpSessionManagerMBean

Managed Bean	Object Name
WebLogicHttpSessionManager	`type=WebLogicHttpSessionManager`, `nodeId=cluster node id`, `appId=web application id`

The `WebLogicHttpSessionManager` class extends the `HttpSessionManager` class. In addition to the information described in Table 6-2, the `WebLogicHttpSessionManager` class also returns the information listed in Table 6-4. Enterprise Manager uses this information to correlate the Coherence*Web instances to the server.

Copyright © 2008, 2026, Oracle and/or its affiliates.

Table 6-4 Information Returned by the WebLogicHttpSessionManager MBean

Attribute Name	Data Type	Description
`ApplicationId`	String	The WebLogic Web application ID.
`ApplicationName`	String	The name of this Web application.
`ApplicationVersion`	String	The version of this Web application.
`DomainName`	String	The WebLogic domain name on which the application is deployed.
`IsEar`	Boolean	Displays `true` if this Web application is a module of an EAR file.
`IsListenAddressEnabled`	Boolean	Displays `true` if a HTTP port is available on this server.
`IsSSLListenPortEnabled`	Boolean	Displays `true` if a HTTPS port is available on this server.
`ListenAddress`	String	The address on which the server is listening.
`ListenPort`	Integer	The port on which this server listens for HTTP requests.
`ServerName`	String	The WebLogic Server name on which the application is deployed.
`SSLListenPort`	Integer	The port on which this server is listening for HTTPS requests.

Running Performance Reports

You can monitor session management performance using performance reports such as the Web Session report, Storage report, Web Session Overflow report, Web report, WebLogic Web report, and Web Service report.
For information about configuring the JMX reporting for Coherence*Web performance reporting, see Using Oracle Coherence Reporting in *Managing Oracle Coherence*.

This section includes the following topics:

- Web Session Cache Storage Report
- Web Session Cache Overflow Report
- Web Report
- WebLogic Web Report
- Web Service Report

Web Session Cache Storage Report

The Web Session Storage report records statistics on the activity between the cluster and the cache where session objects and data are stored. The statistics include information about the number of put, get, and prune operations performed on the session storage cache, and the amount of time spent on these operations.

The report is a tab-delimited file that is prefixed with the date in `YYYYMMDDHH` format and appended with `cache-session-storage.txt`. For example `2010013113-cache-session-storage.txt` would be created on January 31, 2010 1:00 pm. Table 6-5 describes the contents of the Web Session Storage report.

Copyright © 2008, 2026, Oracle and/or its affiliates.

Table 6-5 Contents of the Web Session Cache Storage Report

Column Title	Data Type	Description
`Batch Counter`	long	A sequential counter to help integrate information between related files. This value resets when the reporter restarts and is not consistent across nodes. However, it is helpful when trying to integrate files.
`Cache Name`	String	Always `session-storage`. It is used to maintain consistency with the Cache Utilization report.
`Evictions`	long	The total number of sessions that have been evicted for the cache across the cluster since the last time the report was created.
`Report Time`	Date	The system time when the report was created.
`Service`	String	The service name of the session cache.
`Tier`	String	The value can be either `front` or `back`. Describes whether the cache resides in the front tier (local cache) or back tier (remote cache).
`NodeId`	String	The numeric cluster member identifier.
`Eviction Count`	Long	The total number of evictions for the cache across the cluster since the last report refresh.
`Total Failures`	long	The total number of session storage write failures for the cache across the cluster since the last time the report was created.
`Total Gets`	long	The total number of session get operations across the cluster since the last time the report was created.
`GetsMillis`	long	The total number of milliseconds spent for each `get()` invocation (`GetsMillis`) to get the sessions across the cluster since the last time the report was created.
`Hits`	long	The total number of session hits across the cluster since the last time the report was created.
`HitsMillis`	long	The total number of milliseconds spent for each `get()` invocation that is a hit (`HitsMillis`) for the session storage across the cluster since the last time the report was created.
`Misses`	long	The total number of sessions get operations that returned misses for the cache across the cluster since the last time the report was created.
`MissesMillis`	long	The total number of milliseconds spent for each `get()` invocation that is a miss (`MissesMillis`) for the session storage across the cluster since the last time the report was created.
`Cache Prunes`	long	The total number of times the session storage cache has been pruned across the cluster since the last time the report was created.
`Cache Prunes Millis`	long	The total number of milliseconds spent for the prune operation (`PrunesMillis`) to prune the session storage cache across the cluster since the last time the report was created.

Copyright © 2008, 2026, Oracle and/or its affiliates.

Table 6-5 (Cont.) Contents of the Web Session Cache Storage Report

Column Title	Data Type	Description
`Puts`	long	The total number of session updates (put operations) across the cluster since the last time the report was created.
`PutsMillis`	long	The total number of milliseconds spent for each `put()` invocation (`PutsMillis`) to update sessions across the cluster since the last time the report was created.
`Total Queue`	long	The sum of the queue links for the session storage cache across the cluster.
`Total Writes`	long	The total number of sessions written to an external cache storage for the cache across the cluster since the last time the report was created.
`Total Writes Millis`	long	The total number of milliseconds spent for each write operation (`WritesMillis`) to update an external cache storage across the cluster since the last time the report was created.

Web Session Cache Overflow Report

The Web Session Overflow report records statistics on the activity between the cluster and the cache where the overflow of session objects and data is stored. The statistics include information about the number of put, get, and prune operations performed on the session overflow cache, and the amount of time spent on these operations.

The report is a tab-delimited file that is prefixed with the date in `YYYYMMDDHH` format and appended with `cache-session-overflow.txt`. For example `2010013113-cache-session-overflow.txt` would be created on January 31, 2010 1:00 pm. Table 6-6 describes the contents of the Web Session Overflow report.

Table 6-6 Contents of the Web Session Overflow Report

Column Title	Data Type	Description
`Batch Counter`	long	A sequential counter to help integrate information between related files. This value does reset when the Reporter restarts and is not consistent across nodes. However, it is helpful when trying to integrate files.
`Cache Name`	String	Always `session-overflow`. It is used to maintain consistency with the Cache Utilization report.
`Evictions`	long	The total number of session overflows that have been evicted for the cache across the cluster since the last time the report was created.
`Report Time`	Date	The system time when the report executed.
`Service`	String	The service name of the session cache.
`NodeId`	String	The numeric cluster member identifier.
`Eviction Count`	Long	The total number of evictions for the cache across the cluster since the last report refresh.

Copyright © 2008, 2026, Oracle and/or its affiliates.

Table 6-6 (Cont.) Contents of the Web Session Overflow Report

Column Title	Data Type	Description
`Tier`	String	The value can be either `front` or `back`. Describes whether the cache resides in the front-tier (local cache) or back tier (remote cache).
`Total Failures`	long	The total number of session overflows storage write failures for the cache across the cluster since the last time the report was created.
`Total Gets`	long	The total number of session overflows get operations across the cluster since the last time the report was created.
`GetsMillis`	long	The total number of milliseconds spent for each `get()` invocation (`GetsMillis`) to get the session overflows across the cluster since the last time the report was created.
`Hits`	long	The total number of session overflow hits across the cluster since the last time the report was created.
`HitsMillis`	long	The total number of milliseconds spent for each `get()` invocation that is a hit (`HitsMillis`) for the session overflow across the cluster since the last time the report was created.
`Misses`	long	The total number of session overflow get operations that returned misses for the cache across the cluster since the last time the report was created.
`MissesMillis`	long	The total number of milliseconds spent for each `get()` invocation that is a miss (`MissesMillis`) for the session overflow across the cluster since the last time the report was created.
`Cache Prunes`	long	The total number of times the session overflow cache has been pruned across the cluster since the last time the report was created.
`Cache Prunes Millis`	long	The total number of milliseconds spent for the prune operations (`PrunesMillis`) to prune the session overflow cache across the cluster since the last time the report was created.
`Puts`	long	The total number of session overflows (put operations) across the cluster since the last time the report was created.
`PutsMillis`	long	The total number of milliseconds spent per `put()` invocation (`PutsMillis`) to update session overflows across the cluster since the last time the report was created.
`Total Queue`	long	The sum of the queue link size for the session overflow cache across the cluster.
`Total Writes`	long	The total number of session overflows written to an external cache storage for the cache across the cluster since the last time the report was created.

Copyright © 2008, 2026, Oracle and/or its affiliates.

Table 6-6 (Cont.) Contents of the Web Session Overflow Report

Column Title	Data Type	Description
`Total Writes Millis`	long	The total number of milliseconds spent for each write operation (`WritesMillis`) to update an external session overflow storage across the cluster since the last time the report was created.

Web Report

The Web Report (`report-web.xml`) provides information about Coherence*Web activity for the cluster. The report is a tab-delimited file that is prefixed with the date and hour in `YYYYMMDDHH` format and appended with `-web.txt`. For example `2009013102-web.txt` would be created on January 1, 2009 at 2:00 am. Table 6-7 describes the contents of the Web Report.

Table 6-7 Contents of the Web Report

Column	Data Type	Description
`Application`	String	The application name.
`Batch Counter`	long	A sequential counter to help integrate information between related files. This value does reset when the Reporter restarts and is not consistent across nodes. However, it is helpful when trying to integrate files.
`Current Overflow Updates`	long	The number of overflow updates since the last time the report was created.
`Current Session Updates`	long	The number of session updates since the last time the report was created.
`LocalAttributeCount`	long	The attribute count on the node.
`LocalSessionCount`	long	The session count on the node.
`Node Id`	integer	The node identifier.
`OverflowAvgSize`	float	The average size for attribute overflows.
`OverflowMaxSize`	long	The maximum size for an attribute overflow.
`OverflowUpdates`	long	The total number of attribute overflow updates since the last time statistics were reset.
`Report Time`	Date	The system time when the report was created.
`SessionAverageLifetime`	float	The average number of seconds a session is active.
`SessionAverageSize`	float	The average size for a session.
`SessionMaxSize`	long	The maximum size for a session.
`SessionMinSize`	long	The minimum size for a session.
`SessionStickyCount`	long	The number of sticky sessions on the node.
`SessionUpdateCount`	long	The number of session updates since the last time statistics were reset.

Copyright © 2008, 2026, Oracle and/or its affiliates.

WebLogic Web Report

The Weblogic Web Report (`report-web-weblogic.xml`) provides information on Coherence*Web activity when it is being used in WebLogic Server environments. This report provides the same information as provided by the Web Report with additional columns for the WebLogic Server name and domain name. The report is a tab-delimited file that is prefixed with the date and hour in `YYYYMMDDHH` format and appended with `-web-weblogic.txt`. For example `2009013102-web-weblogic.txt` would be created on January 1, 2009 at 2:00 am.

Table 6-8 Contents of the WebLogic Web Report

Column	Data Type	Description
`Application`	String	The application name.
`Batch Counter`	long	A sequential counter to help integrate information between related files. This value does reset when the Reporter restarts and is not consistent across nodes. However, it is helpful when trying to integrate files.
`Current Overflow Updates`	long	The number of overflow updates since the last time the report was created.
`Current Session Updates`	long	The number of session updates since the last time the report was created.
`DomainName`	String	The name of the WebLogic Server domain in which Coherence*Web is running.
`LocalAttributeCount`	long	The attribute count on the node.
`LocalSessionCount`	long	The session count on the node.
`Node Id`	integer	The node identifier.
`OverflowAvgSize`	float	The average size for attribute overflows.
`OverflowMaxSize`	long	The maximum size for an attribute overflow.
`OverflowUpdates`	long	The total number of attribute overflow updates since the last time statistics were reset.
`Report Time`	Date	The system time when the report was created.
`ServerName`	String	The name of the WebLogic Server on which Coherence*Web is running.
`SessionAverageLifetime`	float	The average number of seconds a session is active.
`SessionAverageSize`	float	The average size for a session.
`SessionMaxSize`	long	The maximum size for a session.
`SessionMinSize`	long	The minimum size for a session.
`SessionStickyCount`	long	The number of sticky sessions on the node.
`SessionUpdateCount`	long	The number of session updates since the last time statistics were reset.

Web Service Report

The Web Service report provides information about the service running the Coherence*Web application. The report records the requests processed, request failures, and request backlog,

Copyright © 2008, 2026, Oracle and/or its affiliates.

tasks processed, task failures, and task backlog. `Request Count` and `Task Count` are useful to determine performance and throughput of the service. `RequestPendingCount` and `Task Backlog` are useful in determining capacity issues or blocked processes. `Task Hung Count`, `Task Timeout Count`, `Thread Abandoned Count`, `Request Timeout Count` are the number of unsuccessful executions that have occurred in the system.

The report is a tab-delimited file that is prefixed with the date and hour in `YYYYMMDDHH` format and appended with `-web-session-service.txt`. For example `2009013102-web-session-service.txt` would be created on January 1, 2009 at 2:00 am. Table 6-9 describes the contents of the Web Service Report.

Table 6-9 Contents of the Web Service Report

Column Title	Data Type	Description
`Batch Counter`	Long	A sequential counter to help integrate information between related files. This value does reset when the Reporter restarts and is not consistent across nodes. However, it is helpful when trying to integrate files.
`Node Id`	String	The numeric node identifier.
`Refresh Time`	Date	The system time when the service information was updated from a remote node.
`Request Count`	Long	The number of requests by the Coherence*Web application since the last report was created.
`RequestPendingCount`	Long	The number of pending requests by the Coherence*Web application at the time of the report.
`RequestPendingDuration`	Long	The duration for the pending requests of the Coherence*Web application at the time of the report.
`Request Timeout Count`	Long	The number of request timeouts by the Coherence*Web application since the last report was created.
`Report Time`	Date	The system time when the report executed.
`Service`	String	A static value (`DistributedSessions`) used as the service name if merging the information with the service file.
`Task Backlog`	Long	The task backlog of the Coherence*Web application at the time of the report was created.
`Task Count`	Long	The number of tasks executed by the Coherence*Web application since the last report was created.
`Task Hung Count`	Long	The number of tasks that were hung by the Coherence*Web application since the last report was created.
`Task Timeout Count`	Long	The number of task timeouts by the Coherence*Web application since the last report was created.
`Thread Abandoned Count`	Long	The number of threads abandoned by the Coherence*Web application since the last report was created.

Copyright © 2008, 2026, Oracle and/or its affiliates.

7

Cleaning Up Expired HTTP Sessions

The Coherence*Web Session Management Module includes a Session Reaper that is responsible for removing HTTP sessions that have expired.

You should monitor the Session Reaper and ensure that it is properly configured based on your application and deployment environment. Properly managing the Session Reaper can help minimize resource usage and can help increase performance.

Each HTTP session contains two pieces of information that determine when it has timed out. The first is the `LastAccessedTime` property of the session, which is the time stamp of the most recent activity involving the session. The second is the `MaxInactiveInterval` property of the session, which specifies how long the session is kept active without any activity; a typical value for this property is 30 minutes. The `MaxInactiveInterval` property defaults to the value configured for Coherence*Web, but it can be modified on a session-by-session basis.

Each time that an HTTP request is received by the server, if there is an HTTP session associated with that request, then the `LastAccessedTime` property of the session is automatically updated to the current time. As long as requests continue to arrive related to that session, it is kept active, but when a period of inactivity occurs longer than that specified by the `MaxInactiveInterval` property, then the session expires. Session expiration is passive—occurring only due to the passing of time. The Coherence*Web Session Reaper scans for sessions that have expired, and when it finds expired sessions it destroys them.

This chapter includes the following sections:

- Understanding the Session Reaper
- Tuning the Session Reaper
- Getting Session Reaper Performance Statistics
- Understanding Session Invalidation Exceptions for the Session Reaper
- Verifying Session Expiry Failover and Cleanup
  Test expiry, invalidation and failover as distinct behaviors before tuning reaper and storage resources.

Understanding the Session Reaper

The Session Reaper configuration addresses basic questions such as how frequently will the reaper run, on which servers will the reaper run, and on which servers will it look for expired sessions.

Understanding Where the Session Reaper Runs

Every application server running Coherence*Web runs the Session Reaper. That means that if Coherence is configured to provide a separate cache tier (made up of cache servers), then the Session Reaper does not run on those cache servers.

By default, the Session Reaper runs concurrently on all of the application servers, so that all of the servers share the workload of identifying and cleaning up expired sessions. The `coherence-reaperdaemon-cluster-coordinated` context parameter causes the cluster to coordinate reaping so that only one server at a time performs the actual reaping; the use of this

Copyright © 2008, 2026, Oracle and/or its affiliates.

option is not suggested, and it cannot be used with the Coherence*Web over Coherence*Extend topology.

The `coherence-reaperdaemon-cluster-coordinated` context parameter should not be used if sticky optimization (`coherence-sticky-sessions`) is also enabled. Because only one server at a time performs the reaping, sessions owned by other nodes cannot be reaped. This means that it will take longer for sessions to be reaped as more nodes are added to the cluster. Also, the reaping ownership does not circulate over the nodes in the cluster in a controlled way; one node can be the reaping node for a long time before it is taken over by another node. During this time, only its own sessions are reaped.

Understanding How Frequently the Session Reaper Runs

The Session Reaper is configured to scan the entire set of sessions over a certain period, called a reaping cycle, which defaults to five minutes. This length of the reaping cycle is specified by the `coherence-reaperdaemon-cycle-seconds` context parameter. This setting indicates to the Session Reaper how aggressively it must work. If the cycle length is configured too short, the Session Reaper uses additional resources without providing additional benefit. If the cycle length is configured too long, then expired sessions will use heap space in the Coherence caches unnecessarily. In most situations, it is far preferable to reduce resource usage than to ensure that sessions are cleaned up quickly after they expire. Consequently, the default cycle of five minutes is a good balance between promptness of cleanup and minimal resource usage.

During the reaping cycle, the Session Reaper scans for expired sessions. In most cases, the Session Reaper takes responsibility for scanning all of the HTTP sessions across the entire cluster, but there is an optimization available for the single tier topology. In the single tier topology, when all of the sessions are being managed by storage-enabled Coherence cluster members that are also running the application server, the session storage is colocated with the application server. Consequently, it is possible for the Session Reaper on each application server to scan only the sessions that are stored locally. This behavior can be enabled by setting the `coherence-reaperdaemon-assume-locality` configuration option to `true`.

Regardless of whether the Session Reaper scans only colocated sessions or all sessions, it does so in a very efficient manner by using these advanced capabilities of the Coherence data grid:

- The Session Reaper delegates the search for expired sessions to the data grid using a custom `ValueExtractor` implementation. This `ValueExtractor` takes advantage of the `BinaryEntry` interface so that it can determine if the session has expired without even deserializing the session. As a result, the selection of expired sessions can be delegated to the data grid just like any other parallel query, and can be executed by storage-enabled Coherence members in a very efficient manner.
- The Session Reaper uses the `com.tangosol.net.partition.PartitionedIterator` class to automatically query on a member-by-member basis, and in a random order that avoids harmonics in large-scale clusters.

Each storage-enabled member can very efficiently scan for any expired sessions, and it has to scan only one time per application server per reaper cycle. The result is a default Session Reaper configuration that works well for application server clusters with one or multiple servers.

Understanding How the Session Reaper Runs

Coherence*Web uses a work manager to retrieve threads to execute the parallel reaping. WebLogic Server defines a default work manager, `wm/CoherenceWorkManager`, which it will attempt to use. If no work manager is defined with that name, it will use the default work manager implemented in Coherence.

Copyright © 2008, 2026, Oracle and/or its affiliates.

The default Coherence work manager implementation uses the `java.util.concurrent.ThreadPoolExecutor` Java API to process the submitted tasks using one of the many threads in the pool. A blocking queue such as `LinkedBlockingQueue`, is used to schedule the waiting tasks in the order of first-in-first out (FIFO).

To use the default WebLogic Server work manager, use the WebLogic Remote Console to create a work manager that is named `wm/CoherenceWorkManager`. Also, add the following `resource-ref` element in the application `web.xml` file:

```
<resource-ref>
   <res-ref-name>wm/CoherenceWorkManager</res-ref-name>
   <res-type>commonj.work.WorkManager</res-type>
   <res-auth>Container</res-auth>
   <res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>
```

To ensure that the Session Reaper does not impact the smooth operation of the application server, it breaks up its work into chunks and schedules that work in a manner that spreads the work across the entire reaping cycle. Because the Session Reaper has to know how much work it must schedule, it maintains statistics on the amount of work that it performed in previous cycles, and uses statistical weighting to ensure that statistics from recent reaping cycles count more heavily. There are several reasons why the Session Reaper breaks up the work in this manner:

- If the Session Reaper consumed a large number of CPU cycles simultaneously, it could cause the application to be less responsive to users. By doing a small portion of the work at a time, the application remains responsive.
- One of the key performance enablers for Coherence*Web is the near-caching feature of Coherence; because the sessions that are expired are accessed through that same near cache to clean them, expiring too many sessions too quickly could cause the cache to evict sessions that are being used on that application server, leading to performance loss.

The Session Reaper performs its job efficiently, even with the default configuration by:

- Delegating as much work as possible to the data grid.
- Delegating work to only one member at a time.
- Enabling the data grid to find expired sessions without deserializing them.
- Restricting the usage of CPU cycles.
- Avoiding cache-thrashing of the near caches that Coherence*Web relies on for performance.

Understanding How the Session Reaper Removes Sessions

The Session Reaper can invalidate sessions either in parallel or serially. By default, it invalidates sessions serially, which may be useful if the application server JVM has a high system load due to a large number of concurrent threads. To invalidate sessions in parallel, set the `coherence-reaperdaemon-parallel` context parameter to `true`.

The Session Reaper deletes sessions that have timed-out. The default behavior is to remove the session after fetching it from the local JVM and calling the `invalidate` method on the HTTP session. However, the session reaper can also be configured to delete sessions remotely using a Coherence entry processor. In this case, the `invalidate` method of the HTTP session and the session listeners are not invoked. Deleting sessions remotely is much faster than the default mechanism but should only be used in applications that do not use session

Copyright © 2008, 2026, Oracle and/or its affiliates.

listeners. To configure the reaper to delete sessions remotely, set the `coherence-session-reaping-mechanism` context parameter to `RemoteDelete`.

Tuning the Session Reaper

There are several Session Reaper configuration properties that can be changed based on how your application is implemented and deployed.
To tune the default Session Reaper configuration:

- If the application is deployed with the in-process topology, then set the `coherence-reaperdaemon-assume-locality` configuration option to `true`.
- Because all of the application servers are responsible for scanning for expired sessions, it is reasonable to increase the `coherence-reaperdaemon-cycle-seconds` configuration option if the cluster is larger than 10 application servers. The larger the number of application servers, the longer the cycle can be; for example, with 200 servers, it would be reasonable to set the length of the reaper cycle as high as 30 minutes (that is, setting the `coherence-reaperdaemon-cycle-seconds` configuration option to 1800).
- If the application does not use session listeners, then set the `coherence-session-reaping-mechanism` context parameter to `RemoteDelete`.
- If you want to set the queue size of the Session Reaper work manager, use the `coherence-reaperdaemon-queue-size` configuration option. If not set, the queue size is unlimited. This option is used only when parallel reaping is enabled and is not applicable for Service Provider Interface (SPI).
- If you want to set the maximum number of threads for the Session Reaper daemon, use the `coherence-reaperdaemon-max-threads` configuration option. The default value for this option is `5`.
- If you want to set the Configuration parameter for minimum number of threads for the Session Reaper daemon, use the `coherence-reaperdaemon-min-threads` configuration option. The default value for this option is `1`.

Getting Session Reaper Performance Statistics

The `HttpSessionManagerMBeanWeb` provides performance statistics for monitoring the average time duration for a reap cycle, the number of sessions reaped, and the time until the next reap cycle.
The following performance statistics are available for the Session Reaper:

- `AverageReapDuration`, which is the average reap duration (the time it takes to complete a reap cycle), in milliseconds, since the statistic was reset
- `LastReapDuration`, which is the time in milliseconds it took for the last reap cycle to finish
- `MaxReapedSessions`, which is the maximum number of sessions reaped in a reap cycle since the statistic was reset
- `NextReapCycle`, which is the time (as a `java.lang.Date` data type) for the next reap cycle
- `ReapedSessions`, which is the number of sessions reaped during the last cycle
- `ReapedSessionsTotal`, which is the number of expired sessions that have been reaped since the statistic was reset

See Managing and Monitoring Applications with JMX.

Copyright © 2008, 2026, Oracle and/or its affiliates.

You can access these attributes in a monitoring tool such as JConsole. However, you must set up the Coherence Clustered JMX Framework before you can access them. See Using JMX to Manage Coherence in *Managing Oracle Coherence*.

Understanding Session Invalidation Exceptions for the Session Reaper

Each Coherence*Web instance has a session reaper that periodically iterates through all of the sessions in the session cache and checks for expired sessions. If multiple Web applications are using a Coherence*Web instance, then a reaper from one Web application can invalidate sessions used in a different application.
Session attribute listeners are registered with the Web application that is reaping expired sessions. The listeners attempt to retrieve the session attribute values during invalidation. If the session attributes are dependent on classes that exist only in the original Web application, then a *class not found* exception is thrown and logged in the Session Reaper. These exceptions will not cause any disruption in the Web application or the application server.

Coherence*Web provides a context parameter, `coherence-session-log-invalidation-exceptions`, to control whether these exceptions are logged. The default value, `true`, allows the exceptions to be logged. If you want to suppress the logging of these exceptions, set this context parameter to `false`.

Verifying Session Expiry Failover and Cleanup

Test expiry, invalidation and failover as distinct behaviors before tuning reaper and storage resources.

Exercise the session lifecycle under load and after failures before changing reaper intervals or storage policy.

- Create a session with representative serializable attributes, update it through the intended locking model and verify another application node sees committed state.
- Expire an idle session and verify eventual removal and expected listener behavior. Test explicit invalidation separately; do not assume reaper scheduling is the same as a browser cookie lifetime.
- Run concurrent requests and long requests through the configured lock timeout. Confirm that a reaper does not cause unbounded request latency or mask application misuse of an invalidated session.
- Fail/restart an application node and a storage member in an isolated test. Verify redundancy, model/metadata compatibility, session availability and cleanup after failover.
- Monitor reaper duration/errors, active sessions, backing/overflow caches, memory and service latency. Distinguish session expiry from generic backing-map eviction and failed deserialization.
- For federation or a recovery design, validate propagation, invalidation and logout across the actual topology; a recovered generic cache entry is not automatically a valid authenticated HTTP session.

Tune using Tuning the Session Reaper and inspect Understanding Session Invalidation Exceptions for the Session Reaper. Use Using the Health Check API for service health and Operating Federation Backlogs and Trust Boundaries for replication lag.

Copyright © 2008, 2026, Oracle and/or its affiliates.

8

Working with JSF and MyFaces Applications

Coherence*Web provides support for Jakarta Server Faces (JSF) and MyFaces applications. JSF is a framework that enables you to build user interfaces for Web applications. MyFaces, from the Apache Software Foundation, provides JSF components that extend the JSF specification. MyFaces components are completely compatible with the JSF 1.1 Reference Implementation or any other compatible implementation.
This chapter includes the following sections:

The framework-specific examples describe the listed JSF/MyFaces integration and do not certify all current framework releases. Select a compatible Jakarta Faces/MyFaces and Servlet/container namespace set, then test actual view/session state serialization, metadata admission, failover and cleanup.

For current preparation use Selecting CoherenceWeb Artifacts and Container APIs and Preparing Session Attributes for Hardened Mode.

- Configuring for all JSF and MyFaces Web Applications:
- Configuring for Instrumented Applications that use MyFaces
- Configuring for Instrumented Applications that use Mojarra

Configuring for all JSF and MyFaces Web Applications:

JSF and MyFaces attempts to cache the state of the view in the session object.
This state data should be serializable by default, but there could be situations where this would not be the case. For example:

- If Coherence*Web reports `IllegalStateException` due to a non-serializable class, and all the attributes placed in the session by your Web-application are serializable, then you must configure JSF/MyFaces to store the state of the view in a hidden field on the rendered page.
- If the Web application puts non-serializable objects in the session object, you must set the `coherence-preserve-attributes` context parameter to `true`.

The JSF parameter `jakarta.faces.STATE_SAVING_METHOD` identifies where the state of the view is stored between requests. By default, the state is saved in the servlet session. Set the `STATE_SAVING_METHOD` parameter to `client` in the `context-param` stanza of the `web.xml` file, so that JSF stores the state of the entire view in a hidden field on the rendered page. If you do not, then JSF may attempt to cache that state, which is not serializable, in the session object.

Example 8-1 illustrates setting the `STATE_SAVING_METHOD` parameter in the `web.xml` file.

Example 8-1 Setting STATE_SAVING_METHOD in the web.xml File

```
...
<context-param>
    <param-name>jakarta.faces.STATE_SAVING_METHOD</param-name>
    <param-value>client</param-value>
</context-param>
...
```


Copyright © 2008, 2026, Oracle and/or its affiliates.

Configuring for Instrumented Applications that use MyFaces

If you are deploying the MyFaces application with the Coherence*Web WebInstaller (that is, an *instrumented* application), then you might have to complete an additional step based on the version of MyFaces.

- If you are using Coherence*Web WebInstaller to deploy a Web-application built with a pre-1.1.*n* version of MyFaces, then nothing more needs to be done.
- If you are using Coherence*Web WebInstaller to deploy a Web-application built with a 1.2.x version of MyFaces, then add the context parameter `org.apache.myfaces.DELEGATE_FACES_SERVLET` to the `web.xml` file. This parameter allows you to specify a custom servlet instead of the default `jakarta.faces.webapp.FacesServlet`.

  Example 8-2 illustrates setting the `DELEGATE_FACES_SERVLET` context parameter in the `web.xml` file.

Example 8-2 Setting DELEGATE_FACES_SERVLET in the web.xml File

```
...
<context-param>
    <param-name>org.apache.myfaces.DELEGATE_FACES_SERVLET</param-name>
    <param-value>com.tangosol.coherence.servlet.api23.ServletWrapper</param-value>
</context-param>
...
```

Configuring for Instrumented Applications that use Mojarra

If you are using Coherence*Web WebInstaller to deploy a Web application based on the JSF Reference Implementation (Mojarra), then you must declare the `FacesServlet` class in the `servlet` stanza of the `web.xml` file.

Example 8-3 Declaring the Faces Servlet in the web.xml File

```
...
<servlet>
     <servlet-name>Faces Servlet (for loading config)</servlet-name>
     <servlet-class>jakarta.faces.webapp.FacesServlet</servlet-class>
 </servlet>
...
```


Copyright © 2008, 2026, Oracle and/or its affiliates.

A

Coherence*Web Context Parameters

Configure Coherence*Web session behavior with the context parameters supported by the selected integration.
For example, the context parameter `coherence-enable-sessioncontext` can be declared on the command line by:

```
-Dcoherence.enable.sessioncontext=true
```

If both a system property and the equivalent context parameter are configured, the value from the system property is honored.

Table A-1 describes the context parameters for Coherence*Web.

Table A-1 Context Parameters for Coherence*Web

Parameter Name	Description
`coherence-application-name`	Coherence*Web uses the value of this parameter to determine the name of the application that uses the `ApplicationScopeController` interface to scope attributes. The value for this parameter should be provided in the following format: *application name* + ! + *Web module name* The *application name* is the name of the application that uses the `ApplicationScopeController` interface and *Web module name* is the name of the Web module in which it appears. For example, if you have an EAR file named `test.ear` and a Web-module named `app1` defined in the EAR file, then the default value for the `coherence-application-name` parameter would be `test!app1`. If this parameter is not configured, then Coherence*Web uses the name of the class loader instead. Also, if the parameter is not configured and the `ApplicationScopeController` interface *is* configured, then a warning is logged saying that the application name was not configured. See Session Attribute Scoping.
`coherence-attribute-overflow-threshold`	For the Split Model, described in Session Models, this value specifies the minimum length (in bytes) that the serialized form of an attribute value must be for it to be stored in the separate overflow cache that is reserved for large attributes. If unspecified, this parameter defaults to `1024`.
`coherence-cache-configuration-path`	Specifies the name of the file that Coherence*Web should use to obtain session cache information, instead of using the default `default-session-cache-config.xml` file. See Customizing the Name of the Session Cache Configuration File.
`coherence-cache-delegator-class`	Specifies a cache delegator class that is responsible for manipulating (getting, putting, or deleting) data in the distributed cache. Valid value is: • `com.tangosol.coherence.servlet.LocalSessionCacheDelegator`—This class indicates that the local cache should be used for storing and retrieving the session instance before attempting to use the distributed cache. See Getting Concurrent Access to the Same Session Instance.

Copyright © 2008, 2026, Oracle and/or its affiliates.

Table A-1 (Cont.) Context Parameters for Coherence*Web

Parameter Name	Description
`coherence-cluster-owned`	If `true`, Coherence*Web automatically shuts down the Coherence node when the Web application shuts down. You must use the WAR-scoped cluster node deployment model in this case. See WAR-Scoped Cluster Nodes. If `false`, the Web application is responsible for shutting down the Coherence node (see `com.tangosol.net.CacheFactory.shutdown()` in the Javadoc). You must carefully consider a cluster node-scoping deployment model in this case and the circumstances under which the application shuts down the Coherence node and the side effects of doing so. See Cluster Node Isolation. **Note:** When using the WebInstaller, a value of `true` instructs the WebInstaller to place the Coherence library in the `WEB-INF/lib` directory of *each* Web application found in your Jakarta EE application. If unspecified, this parameter defaults to `false`.
`coherence-configuration-consistency`	If `true`, Coherence*Web runs a configuration check at startup to determine whether all nodes in the Web tier have the same Coherence*Web configuration. If the configuration of a particular node is not consistent, then it will fail to start (which, in turn, prevents the application from starting). If `false`, (there is no checking) and the configurations are not consistent, then the cluster members might exhibit inconsistent behavior in managing the session data. If unspecified, this parameter defaults to `false`.
`coherence-contextless-session-retain-millis`	Specifies the number of milliseconds that a server holds a lock on a session while accessing it without the session being implied by the current request context. A session is implied by the current request context if and only if the current thread is processing a servlet request, and the request is associated with that session. All other access to a session object is out of context. For example, if a reference to an arbitrary session is obtained from a `SessionContext` object (if that option is enabled), or if the application has code that holds on to session object references to manage sessions directly. Because session access requires session ownership, out of context access to the session object automatically obtains ownership on behalf of the caller; that ownership will be retained for the number of milliseconds specified by this option so that repeated calls to the session do not individually obtain and release ownership, which is potentially an expensive operation. The valid range is `10` to `10000` (from `1/100`th of a second up to `10` seconds). If unspecified, this parameter defaults to `200`.

Copyright © 2008, 2026, Oracle and/or its affiliates.

Table A-1 (Cont.) Context Parameters for Coherence*Web

Parameter Name	Description
`coherence-distributioncontroller-class`	This value specifies a class name of the `com.tangosol.coherence.servlet.HttpSessionCollection$SessionDistributionController` interface implementation. Valid values include: • `com.tangosol.coherence.servlet.AbstractHttpSessionCollection$DistributedController` an implementation of the `SessionDistributionController` interface that forces all sessions (and thus their attributes) to be managed in a distributed manner. This is the default behavior, but by having an implementation that forces this, the raw overhead of using a `HttpSessionController` can be measured. • `com.tangosol.coherence.servlet.AbstractHttpSessionCollection$HybridController` an implementation of the `SessionDistributionController` interface that forces all sessions and serializable attributes to be managed in a distributed manner. All session attributes that do not implement the `Serializable` interface will be kept local. • `com.tangosol.coherence.servlet.AbstractHttpSessionCollection$LocalController` an implementation of the `SessionDistributionController` interface that forces all sessions (and thus their attributes) to be managed locally. This might not be useful for production purposes, but it can be useful for testing the difference in scalable performance between local-only and fully-distributed implementations.
`coherence-enable-sessioncontext`	When set to `true`, this parameter allows the application to iterate sessions from the session context, thus disobeying the deprecation in the servlet specification. If unspecified, this parameter defaults to `false`.
`coherence-eventlisteners`	This is the comma-delimited list of names of application classes that want to receive events from the Web container. This list comes from the application listeners declared in the `listener` elements of the `web.xml` file.
`coherence-enable-suspect-attributes`	If set to `true`, Coherence*Web attempts to detect whether the value of any session-related attributes have changed. Attributes that can be changed (determined with a simple check) and that can be accessed by a `get` method are deemed to be suspect. Changeable objects might have been changed by application code and must be re-serialized back into the cache. See Detecting Changed Attribute Values. If unspecified, this parameter defaults to `true`.
`coherence-factory-class`	This is the fully qualified name of the class that implements the `SessionHelper.Factory` factory class. This parameter defaults to `com.tangosol.coherence.servlet.api`*nn*`.DefaultFactory` where *nn* is `50`, `60`, or `61` for Servlet 5.0, 6.0, or 6.1 containers, respectively. Use 6.1 if your Servlet container is later than 6.1.
`coherence-local-session-cachename`	This name overrides the name of the local cache that stores nondistributed sessions when the `coherence-distributioncontroller-class` parameter is specified. If unspecified, this parameter defaults to `local-session-storage`. See Session Cache Configuration File.

Copyright © 2008, 2026, Oracle and/or its affiliates.

Table A-1 (Cont.) Context Parameters for Coherence*Web

Parameter Name	Description
`coherence-local-attribute-cachename`	This name overrides the name of the local cache that stores non-distributed sessions when either the `coherence-sessiondistributioncontroller-class` parameter is specified or the `coherence-preserve-attributes` parameter is `true`. If unspecified, this parameter defaults to `local-attribute-storage`. See Session Cache Configuration File.
`coherence-preserve-attributes`	This value, if set to `true`, specifies that non-serializable attributes should be preserved as local ones. This parameter requires a load balancer to be present to retrieve non-serializable attributes for a session. These attributes will be lost if the client (application server) fails. The application would need to be able to recover from this. If unspecified, this parameter defaults to `false`.
`coherence-reaperdaemon-assume-locality`	This setting allows the Session Reaper to assume that the sessions that are stored on this node (for example, by a distributed cache service) are the only sessions that this node must check for expiration. This value must be set to `false` if the session storage cache is being managed by nodes that are not running a reaper, for example if cache servers are being used to manage the session storage cache. If cache servers are being used, select the Split Model and run the session overflow storage in a separate distributed cache service that is managed entirely by the cache servers. Leave the session storage cache itself in a distributed cache service that is managed entirely by the application server JVMs so they can take advantage of this assume locality feature. See Cleaning Up Expired HTTP Sessions . If unspecified, this parameter defaults to `true`.
`coherence-reaperdaemon-cluster-coordinated`	If `true`, the Session Reaper coordinates reaping in the cluster such that only one server will perform reaping within a given reaping cycle, and it will be responsible for checking all of the sessions that are being managed in the cluster. See Cleaning Up Expired HTTP Sessions . This option should not be used if sticky optimization (`coherence-sticky-sessions`) is also enabled. See Understanding the Session Reaper. If unspecified, this parameter defaults to `false`.
`coherence-reaperdaemon-cycle-seconds`	This is the number of seconds that the daemon rests between reaping. For production clusters with long session timeout intervals, this can safely be set higher. For testing, particularly with short session timeout intervals, it can be set much lower. Setting it too low can cause more network traffic and use more processing cycles, and has benefit only if the application requires the sessions to be invalidated quickly when they have expired. See Cleaning Up Expired HTTP Sessions . If unspecified, this parameter defaults to `300`.
`coherence-reaperdaemon-parallel`	If set to `true`, the Session Reaper will invalidate expired sessions in parallel. When set to `false`, expired sessions will be invalidated serially. See Understanding the Session Reaper. The default is `false`.
`coherence-reaperdaemon-min-threads`	Configuration parameter for setting the minimum number of threads for the session reaper daemon. If unspecified, this parameter defaults to `1`.

Copyright © 2008, 2026, Oracle and/or its affiliates.

Table A-1 (Cont.) Context Parameters for Coherence*Web

Parameter Name	Description
`coherence-reaperdaemon-max-threads`	Configuration parameter for setting the maximum number of threads for the session reaper daemon. If unspecified, this parameter defaults to 5.
`coherence-reaperdaemon-priority`	This is the priority for the Session Reaper daemon. See Cleaning Up Expired HTTP Sessions and the source for the `java.lang.Thread` class. If unspecified, this parameter defaults to 5.
`coherence-reaperdaemon-sweep-modulo`	The number of times that the reaper reaps the sessions that are being used locally before it will check sessions that may be orphaned or expired elsewhere in the cluster. If unspecified, this parameter defaults to 4.
`coherence-session-reaping-mechanism`	This property indicates the mechanism that is used by the session reaper to delete timed-out sessions. Valid values are `Default` and `RemoteDelete`. See Cleaning Up Expired HTTP Sessions . The default is `Default`
`coherence-scopecontroller-class`	This value specifies a class name of the optional `com.tangosol.coherence.servlet.HttpSessionCollection$AttributeScopeController` interface implementation. See Session Attribute Scoping. Valid values include: • `com.tangosol.coherence.servlet.AbstractHttpSessionCollection$ApplicationScopeController` • `com.tangosol.coherence.servlet.AbstractHttpSessionCollection$GlobalScopeController` The default value for Coherence*Web is `com.tangosol.coherence.servlet.AbstractHttpSessionCollection$ApplicationScopeController` For Coherence*Web WebInstaller, there is no declared default value.
`coherence-servletcontext-clustered`	This value is either `true` or `false` to indicate whether the attributes of the `ServletContext` will be clustered. If `true`, then all serializable `ServletContext` attribute values will be shared among all cluster nodes. If unspecified, this parameter defaults to `false`, primarily because the servlet specification indicates that the `ServletContext` attributes are local to a JVM and should not be clustered.
`coherence-servletcontext-cachename`	This specifies the name of the Coherence cache to be used to hold the servlet context data if the servlet context is clustered. If unspecified, this parameter defaults to `servletcontext-storage`. SeeSession Cache Configuration File.

Copyright © 2008, 2026, Oracle and/or its affiliates.

Table A-1 (Cont.) Context Parameters for Coherence*Web

Parameter Name	Description

`coherence-session-affinity-token`	Configures the session affinity suffix token with a given value. For example, to set the session affinity suffix to `abcd`, add the following code to the Web application's `web.xml` file:
	
	`<context-param>`
	`  <param-name>coherence-session-affinity-token</param-name>`
	`  <param-value>abcd</param-value>`
	`</context-param>`
	
	To strip the session affinity suffix from the token, enter an exclamation point (`!`) as the parameter value. See Sharing Coherence*Web Sessions with Other Application Servers.

`coherence-session-app-locking`	This value, if set to `true`, will prevent two threads in different applications from processing a request for the same session at the same time. If set to `true` the value of the `coherence-session-member-locking` parameter will be ignored, because application locking implies member locking. A value of `false` is incompatible with thread locking. If unspecified, this parameter defaults to `false`. See also coherence-session-member-locking, coherence-session-locking, and coherence-session-app-locking parameter descriptions in this table and Session Locking Modes.
`coherence-session-cachename`	This name overrides the name of the clustered cache that stores the sessions. If unspecified, this parameter defaults to `session-storage`. SeeSession Cache Configuration File.
`coherence-session-cache-federated`	This specifies whether the session cache is federated among cluster participants. Valid values are `true` and `false`. If set to `true`, the `default-federated-session-cache-config.xml` session cache configuration file is used and results in the session cache being federated. See Federated Session Caches. The default is `false`
`coherence-session-cookie-domain`	This specifies the domain of the session cookie as defined by *Request for Comments 2109: HTTP State Management Mechanism* (RFC 2109). By default, no domain is set explicitly by the session management implementation. See Session and Session Attribute Scoping.
`coherence-session-cookie-httponly`	Appends the `HttpOnly` attribute to the session cookie. Note that not all browsers support this functionality. This context parameter can be used only with instrumented applications. See Preventing Cross-Site Scripting Attacks.
`coherence-session-cookie-name`	This specifies the name of the session cookie. If unspecified, this parameter defaults to `JSESSIONID`.
`coherence-session-cookie-path`	This specifies the path of the session cookie as defined by RFC 2109. By default, no path is set explicitly by the session management implementation. See Session and Session Attribute Scoping.
`coherence-session-cookie-max-age`	This specifies the maximum age in seconds of the session cookie as defined by RFC 2109. A value of -1 indicates that the cookie will not persist on the client; a positive value gives the maximum age that the cookie will be persistent for the client. Zero is not permitted. If unspecified, this parameter defaults to -1.

Copyright © 2008, 2026, Oracle and/or its affiliates.

Table A-1 (Cont.) Context Parameters for Coherence*Web

Parameter Name	Description
`coherence-session-cookie-secure`	If `true`, this value ensures that the session cookie will be sent only from a Web client over a TLS connection. If unspecified, the default is `false`.
`coherence-session-cookies-enabled`	If unspecified, this parameter defaults to `true` to enable session cookies.
`coherence-session-expire-seconds`	This value overrides the session expiration time, and is expressed in seconds. Setting it to -1 causes sessions to never expire. See Cleaning Up Expired HTTP Sessions . If unspecified, this parameter defaults to `1800`.
`coherence-session-get-lock-timeout`	This value configures a timeout for lock acquisition for Coherence*Web. See Troubleshooting Locking in HTTP Sessions.
`coherence-session-id-length`	This is the length, in characters, of generated session IDs. The suggested absolute minimum length is 8. If unspecified, this parameter defaults to 12.
`coherence-session-lazy-access`	This value enables lazy acquisition of sessions. A session will be acquired only when the servlet or filter attempts to access it. This is relevant only for instrumented Web applications. See Accessing Sessions with Lazy Acquisition. If unspecified, this parameter defaults to `false`.
`coherence-session-locking`	If `false`, concurrent modification to sessions, with the last update being saved, will be allowed. If `coherence-session-app-locking`, `coherence-session-member-locking`, or `coherence-session-thread-locking` are set to `true`, then this value is ignored (being logically `true`). See Optimistic Locking and Last-Write-Wins Locking. If unspecified, this parameter defaults to `false`. See also coherence-session-app-locking , coherence-session-member-locking, and coherence-session-thread-locking in this table.

Copyright © 2008, 2026, Oracle and/or its affiliates.

Table A-1 (Cont.) Context Parameters for Coherence*Web

Parameter Name	Description

`coherence-session-locking-mode`	The value of this context parameter determines the locking mode that will govern concurrent access to HTTP sessions.
	• `none`—This value allows concurrent access to a session by multiple threads in a single member or multiple members. In this case, the last write is saved. This is the default locking mode. See Last-Write-Wins Locking.
	• `optimistic`—This value allows multiple web container threads in one or more members to access the same session concurrently. See Optimistic Locking.
	• `app`—This value prevent two threads in different applications from processing a request for the same session at the same time. If this parameter is set to `app`, then the value of the `coherence-session-member-locking` parameter will be ignored, because application locking implies member locking. A value of `false` is incompatible with thread locking. See Application Locking .
	• `member`—This value allows multiple web container threads in the same cluster node to access and modify the same session concurrently, but prohibits concurrent access by threads in different members. See Member Locking.
	• `thread`—This value prevents two threads in the same JVM from processing a request for the same session at the same time. If set to `true`, the value of the `coherence-session-member-locking` parameter is ignored, because thread locking implies member locking. See Thread Locking.
	For example, to set the `coherence-session-locking-mode` context parameter to application locking in `web.xml`:
	`<context-param>`
	`<param-name>coherence-session-locking-mode</param-name>`
	`<param-value>app</param-value>`
	`</context-param>`

`coherence-session-log-invalidation-exceptions`	During session invalidation, many *class not found* exceptions might be thrown and logged in the session reaper. If this context parameter is set to `false`, then the exceptions will be suppressed. If set to `true`, then the exceptions will be logged. If unspecified, this parameter defaults to `true`. See Understanding Session Invalidation Exceptions for the Session Reaper.
`coherence-session-log-threads-holding-lock`	If `true`, this value specifies if a diagnostic invocation service is executed when a member cannot acquire the cluster lock for a session. The invocation service will cause the member that has ownership of the session to log the stack trace of the threads that are currently holding the lock. The `coherence-session-log-threads-holding-lock` context parameter is available only when the `coherence-sticky-sessions` context parameter is set to `true`. If unspecified, this parameter defaults to `true`. See Troubleshooting Locking in HTTP Sessions.

Copyright © 2008, 2026, Oracle and/or its affiliates.

Table A-1 (Cont.) Context Parameters for Coherence*Web

Parameter Name	Description
`coherence-session-logger-level`	An alternative way to set the logging level for Coherence*Web (as opposed to JDK logging). The valid values for this parameter are the same as for JDK logging: `SEVERE`, `WARNING`, `INFO`, `CONFIG`, `FINE`, `FINER` (default), and `FINEST`. See Configuring Logging for Coherence*Web. See `java.util.logging` Level class in the Java SE and JDK API Specification.
`coherence-session-management-cachename`	This name overrides the name of the clustered cache that stores the management and configuration information for the session management implementation. If unspecified, this parameter defaults to `session-management`. See Session Cache Configuration File.
`coherence-session-member-locking`	If `true`, this value prevents two threads in different members from processing a request for the same session at the same time. If unspecified, this parameter defaults to `false`. See also coherence-session-thread-locking, coherence-session-locking, and coherence-session-app-locking in this table.
`coherence.session.optimizeModifiedSessions`	This JVM system property, if set to `true`, enables near cache optimizations which can improve performance with applications that use Last-Write-Wins locking. If unspecified, this value defaults to `false`. This parameter can be set only on the command line as a system property.
`coherence-session-overflow-cachename`	For the Split Model, this value overrides the name of the clustered cache that stores the large attributes that exceed a certain size and thus are determined to be more efficiently managed as separate cache entries and not as part of the serialized session object itself. If unspecified, this parameter defaults to `session-overflow`. SeeSession Cache Configuration File.
`coherence-session-strict-spec`	If `false`, then the implementation will not be required to adhere to the servlet specification. The implementation will ignore certain types of exceptions and the application will not terminate. Setting, getting, and removing attributes, or invalidating sessions will not generate any callbacks to session listeners. Any `ClassNotFound` exceptions will not be propagated back to the caller if an attribute cannot be deserialized because the class does not exist in the invoking application. If `true`, then the implementation strictly adheres to the servlet specification. `ClassNotFound` exceptions must be handled by the application, and session listener events will be sent, even if retrieving the attribute value fails. If unspecified, this parameter defaults to `true`.
`coherence-session-thread-locking`	If `true`, this value prevents two threads in the same JVM from processing a request for the same session at the same time. If set to `true`, the value of the `coherence-session-member-locking` parameter is ignored, because thread locking implies member locking. If unspecified, this parameter defaults to `false`. See also coherence-session-app-locking, coherence-session-locking, and coherence-session-member-locking parameter descriptions in this table and Session Locking Modes.

Copyright © 2008, 2026, Oracle and/or its affiliates.

Table A-1 (Cont.) Context Parameters for Coherence*Web

Parameter Name	Description
`coherence-session-urldecode-bycontainer`	If `true`, this value uses the container's decoding of the URL session ID. If `coherence-session-urlencode-name` has been overridden, this must be set to `false`. Setting this to `false` will not work in some containers. If unspecified, this parameter defaults to `true`.
`coherence-session-urlencode-bycontainer`	If `true`, this value uses the container's encoding of the URL session ID. Setting this to `true` could conflict with the setting for `coherence-session-urlencode-name` if it has been specified. If unspecified, this parameter defaults to `false`.
`coherence-session-urlencode-enabled`	If `true`, this value enables URL encoding of session IDs. If unspecified, this parameter defaults to `true`.
`coherence-session-urlencode-name`	This is the parameter name to encode the session ID into the URL. On some containers, this value cannot be overridden. If unspecified, this parameter defaults to `jsessionid`.
`coherence-session-weblogic-compatibility-mode`	If `true`, a single session ID (with the cookie path set to "/") will map to a unique Coherence*Web session instance in each Web application. If `false`, then the standard behavior will apply: that is, a single session ID will map to a single session instance. All other session persistence mechanisms in WebLogic Server use a single session ID in each Web application to refer to different session instances. If unspecified, this parameter defaults to `true`. An exception is when the application is configured to use the global scope controller. In this case, the default is `false`. See Scoping the Session Cookie Path.
`coherence-sessioncollection-class`	This is the fully-qualified class name of the `HttpSessionCollection` implementation to use. Possible values include: • `com.tangosol.coherence.servlet.MonolithicHttpSessionCollection` • `com.tangosol.coherence.servlet.SplitHttpSessionCollection` (default) • `com.tangosol.coherence.servlet.TraditionalHttpSessionCollection` A value must be specified for this parameter. See Configuring a Session Model.
`coherence-shutdown-delay-seconds`	This value determines how long the session management implementation waits before shutting down after receiving the last indication that the application has been stopped, either from `ServletContextListener` events (Servlet 2.3 or later) or by the destruction of `Servlet` and `Filter` objects. This value is expressed in seconds. A value of zero indicates synchronous shutdown; any positive value indicates asynchronous shutdown. If unspecified, this parameter defaults to 0, because some servers are not capable of asynchronous shutdown.

Copyright © 2008, 2026, Oracle and/or its affiliates.

Table A-1 (Cont.) Context Parameters for Coherence*Web

Parameter Name	Description
`coherence-sticky-sessions`	If `true`, this value specifies whether sticky session optimizations will be used. This should be enabled only if a sticky load balancer is being used. This feature requires member, application or thread locking to be enabled. See Enabling Sticky Session Optimizations . See also coherence-session-thread-locking, coherence-session-member-locking, and coherence-session-app-locking in this table. If unspecified, this parameter defaults to `false`.

Copyright © 2008, 2026, Oracle and/or its affiliates.

B

Capacity Planning

Size distributed session storage from measured session graphs, concurrency, backups and operational reserve.

To find the number of cache servers that you will need, you must first calculate the application's heap requirements and the cache server's available tenured generation.

1. Calculate your application's total heap requirements.

   When trying to determine the number of cache servers that you will need for your application, a good starting point is to determine your application's total heap requirements. The total heap requirement can be calculated as the number of sessions that you will run, multiplied by the average number of cached objects per session, multiplied by average number of bytes per cached object. Because you typically make one backup copy per cache entry, multiply the total by 2. Written as an equation, this becomes:

   ```
   Total_Heap_Requirement = 2 * (Number_of_Sessions) *
   (Average_Number_of_Cached_Objects per Session) * (Average_Number_of_Bytes per
   Cached_Object)
   ```

   The units of measure for `Total_Heap_Requirement` are bytes. The `Average_Number_of_Bytes per Cached_Object`, means the number of bytes in the serialized byte stream of primary copies only. Note that this equation does not address unserialized object size. Space requirements for backup copies are accounted for separately.

2. Calculate the available tenured generation in a cache server JVM.

   The available tenured generation is a function of the maximum heap size allocation and other user-specified JVM heap-sizing parameters. Another factor in the available tenured generation is the percentage of the heap that is available for storage. Typically, 66% is used as the maximum percentage of the heap available for storage, but this figure might be too low for your system. Make it a variable:

   ```
   Percent_of_Heap_Available_for_Storage = 0.66

   Available_Tenured_Generation = (Maximum_Heap_Size) *
   (Percent_of_Heap_Available_for_Storage)
   ```

3. Calculate the number of cache servers that will be needed.

   To calculate the number of cache servers that will be needed, divide the total heap requirement by the available tenured generation.

   ```
   Number_of_Cache_Servers = (Total_Heap_Requirement / Available_Tenured_Generation)
   ```

The formulas below are historical illustrations with simplifying assumptions. Use Sizing Modern Session Storage to choose current deployment capacity; neither the two-copy factor nor the 0.66 heap fraction is a required default.

- Sizing Modern Session Storage
  Use workload measurements and failure reserve instead of fixed heap fractions or one session-size average.

Copyright © 2008, 2026, Oracle and/or its affiliates.

Sizing Modern Session Storage

Use workload measurements and failure reserve instead of fixed heap fractions or one session-size average.

Measure actual serialized session graphs, attribute update rates, backups and local front/overflow caches. Include application framework state, indexes, service queues, serialization buffers and JVM/native overhead.

- Start from a representative distribution of session sizes and concurrency, not only the mean. Measure p95/p99 large attributes and short bursts.
- Account for the configured number of primary/backup copies, storage-member failure/redistribution reserve and the selected backing-map implementation. The historical factor of two assumes one primary plus one backup; it is not universal.
- Measure the selected Java 21 collector and heap/native use under session creation, updates, failover and reaper catch-up. A fixed 66 percent heap-storage fraction is an illustrative older sizing assumption, not a modern collector requirement.
- Set alerts for cache/overflow growth, GC pauses, allocation, network, storage and reaper backlog. Re-test the largest expected outage/catch-up window.
- Avoid eviction as a substitute for an intentional session lifetime policy. Validate the user-visible behavior of any configured memory ceiling or serialization fallback.

For service/storage resource tuning use Performance Tuning.

Copyright © 2008, 2026, Oracle and/or its affiliates.

C

Session Cache Configuration File

Coherence*Web uses the session cache configuration file, `default-session-cache-config.xml`, to define the caches and services that implement HTTP session management. This file is deployed in the `WEB-INF/classes` directory.
Table C-1 describes the default cache-related values used in the `default-session-cache-config.xml` file.

Table C-1 Cache-Related Values Used in `default-session-cache-config.xml`

Value	Description
`local-attribute-storage`	This local cache is used to store attributes that are not distributed. This can happen under these conditions: • A `coherence-distributioncontroller-class` is configured. Attributes for local sessions will be stored in this cache. • A non-serializable attribute is set on a distributed session. If `coherence-preserve-attributes` is set to `true`, then non-serializable attributes will be placed in the cache. See Table A-1.
`local-session-storage`	This local cache is used to store session models that are considered to be local by the configured (if any) `coherence-distributioncontroller-class` parameter. See Table A-1.
`servletcontext-storage`	If `ServletContext` attribute clustering (see the `coherence-servletcontext-clustered` parameter in Table A-1) is enabled (it is disabled by default), this cache is used to store `ServletContext` attributes. This cache is expected to contain a few read-mostly attributes.
`session-management`	This cache is used to store internal configuration and management information for the session management implementation. This information is updated infrequently.
`session-storage`	This value is the default clustered cache used to store the session attributes. To use a different cache, use the `coherence-session-cachename` context parameter to specify the cache name.
`session-overflow`	If the `coherence-sessioncollection-class` parameter (described in Table A-1) is set to `com.tangosol.coherence.servlet.SplitHttpSessionCollection`, then this cache will hold large session attributes. By default, session attributes larger than 1 K will be stored in this cache. This is configured as a distributed cache.
`session-storage-heap-only`	This clustered (non-elastic) cache is used to store the session attributes. This value can be put to use with the `coherence-session-cachename` context parameter.
`session-overflow-heap-only`	This clustered (non-elastic) cache is used to store the "overflowing" (split-out due to size) session attributes. It is used only for the "Split" model and can be put to use with the `coherence-session-overflow-cachename` context parameter.

Table C-2 describes the services-related values used in the `default-session-cache-config.xml` file.

Copyright © 2008, 2026, Oracle and/or its affiliates.

Table C-2 Services-Related Values Used in default-session-cache-config.xml

Value	Description
`DistributedSessions`	This distributed service uses the RAM journal backing map (elastic data) and is used by the following caches: • `session-storage` • `session-overflow` The `coherence.session.localstorage` system property controls if a JVM stores and manages data for these caches. Under most circumstances, this should be set to `false` for Web container JVMs. See Deployment Topologies.
`ViewSessionsMisc`	This view cache service.
`SessionOwnership`	This invocation service is used by the sticky session optimization feature (if `coherence-sticky-sessions` is set to `true`).
`DistributedSessionsHeapOnly`	This distributed service is used by the following caches: • `session-storage-heap-only` • `session-overflow-heap-only`

Example C-1 illustrates the contents of the `default-session-cache-config.xml` file. The cache- and services-related values described in Table C-1 and Table C-2 appear in **bold**.

Example C-1 Contents of the `default-session-cache-config.xml` File

```
<?xml version="1.0"?>
<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<!--                                                                    -->
<!--      Cache configuration descriptor for Coherence*Web              -->
<!--                                                                    -->
<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - --
>

<cache-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xmlns="http://xmlns.oracle.com/coherence/coherence-cache-config"
              xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-
cache-config coherence-cache-config.xsd">

  <scope-name>oracle.coherence.web</scope-name>

  <caching-scheme-mapping>
    <!--
    The clustered cache used to store Session management data.
    -->
    <cache-mapping>
      <cache-name>session-management</cache-name>
      <scheme-name>view</scheme-name>
    </cache-mapping>

    <!--
    The clustered cache used to store ServletContext attributes.
    -->
    <cache-mapping>
      <cache-name>servletcontext-storage</cache-name>
      <scheme-name>view</scheme-name>
    </cache-mapping>
```


Copyright © 2008, 2026, Oracle and/or its affiliates.

```
    <!--
    The clustered cache used to store Session attributes.
    -->
    <cache-mapping>
      <cache-name>session-storage</cache-name>
      <scheme-name>session-distributed</scheme-name>
    </cache-mapping>

    <!--
    The clustered (non-elastic) cache used to store Session attributes.
    -->
    <cache-mapping>
      <cache-name>session-storage-heap-only</cache-name>
      <scheme-name>session-distributed-heap-only</scheme-name>
    </cache-mapping>

    <!--
    The clustered cache used to store the "overflowing" (split-out due to
size)
    Session attributes. Only used for the "Split" model.
    -->
    <cache-mapping>
      <cache-name>session-overflow</cache-name>
      <scheme-name>session-distributed</scheme-name>
    </cache-mapping>

    <!--
    The clustered (non-elastic) cache used to store the "overflowing" (split-
out due to size)
    Session attributes. Only used for the "Split" model.
    -->
    <cache-mapping>
      <cache-name>session-overflow-heap-only</cache-name>
      <scheme-name>session-distributed-heap-only</scheme-name>
    </cache-mapping>

    <!--
    The local cache used to store Sessions that are not yet distributed (if
    there is a distribution controller).
    -->
    <cache-mapping>
      <cache-name>local-session-storage</cache-name>
      <scheme-name>unlimited-local</scheme-name>
    </cache-mapping>

    <!--
    The local cache used to store Session attributes that are not distributed
    (if there is a distribution controller or attributes are allowed to become
    local when serialization fails).
    -->
    <cache-mapping>
      <cache-name>local-attribute-storage</cache-name>
      <scheme-name>unlimited-local</scheme-name>
    </cache-mapping>
  </caching-scheme-mapping>
```


Copyright © 2008, 2026, Oracle and/or its affiliates.

```
  <caching-schemes>
    <!--
    View caching scheme used by the Session management and ServletContext
    attribute caches.
    -->
    <view-scheme>
      <scheme-name>view</scheme-name>
      <back-scheme>
        <distributed-scheme>
          <scheme-ref>session-distributed-heap-only</scheme-ref>
        </distributed-scheme>
      </back-scheme>
      <reconnect-interval>30s</reconnect-interval>
      <autostart>true</autostart>
    </view-scheme>

    <local-scheme>
      <scheme-name>session-front</scheme-name>
      <eviction-policy>HYBRID</eviction-policy>
      <high-units>1000</high-units>
      <low-units>750</low-units>
    </local-scheme>

    <distributed-scheme>
      <scheme-name>session-distributed</scheme-name>
      <service-name>DistributedSessions</service-name>
      <lease-granularity>member</lease-granularity>
      <local-storage system-property="coherence.session.localstorage">false</
local-storage>
      <partition-count>257</partition-count>
      <backup-count>1</backup-count>
      <request-timeout>30s</request-timeout>
      <backing-map-scheme>
        <ramjournal-scheme>
           <high-units system-property="coherence.session.highunits"/>
           <unit-calculator>BINARY</unit-calculator>
        </ramjournal-scheme>
      </backing-map-scheme>
      <autostart>true</autostart>
    </distributed-scheme>

    <distributed-scheme>
      <scheme-name>session-distributed-heap-only</scheme-name>
      <service-name>DistributedSessionsHeapOnly</service-name>
      <lease-granularity>member</lease-granularity>
      <local-storage system-property="coherence.session.localstorage">false</
local-storage>
      <partition-count>257</partition-count>
      <backup-count>1</backup-count>
      <request-timeout>30s</request-timeout>
      <backing-map-scheme>
        <local-scheme/>
      </backing-map-scheme>
      <autostart>true</autostart>
    </distributed-scheme>
```


Copyright © 2008, 2026, Oracle and/or its affiliates.

```
    <!--
    Local caching scheme definition used by all caches that do not require an
    eviction policy.
    -->
    <local-scheme>
      <scheme-name>unlimited-local</scheme-name>
      <service-name>LocalSessionCache</service-name>
    </local-scheme>

    <!--
    Clustered invocation service that manages sticky session ownership.
    -->
    <invocation-scheme>
      <service-name>SessionOwnership</service-name>
      <request-timeout>30s</request-timeout>
    </invocation-scheme>
  </caching-schemes>
</cache-config>
```

Use the default session cache configuration shipped with the exact selected Coherence*Web module as the starting point for customization. The ramjournal-scheme in this example is Elastic Data Journal. It is not Unified Persistence Journal, and adding generic persistence does not by itself establish supported HTTP session lifecycle/recovery semantics. Keep application cache/persistence configuration distinct and qualify session invalidation, expiry and class evolution with the selected deployment.

Copyright © 2008, 2026, Oracle and/or its affiliates.