# Red Hat JBoss Enterprise Application Platform 7.2

## Migration Guide

For Use with Red Hat JBoss Enterprise Application Platform 7.2

Last Updated: 2019-09-26

# Red Hat JBoss Enterprise Application Platform 7.2 Migration Guide

For Use with Red Hat JBoss Enterprise Application Platform 7.2

## Legal Notice

## Abstract

This guide provides information about how to migrate your application from previous versions of Red Hat JBoss Enterprise Application Platform.

# Table of Contents

# CHAPTER 1. INTRODUCTION

This guide documents the changes required to successfully run and deploy Red Hat JBoss Enterprise Application Platform 6 applications on Red Hat JBoss Enterprise Application Platform 7. It provides information about the new features available in this release, the deprecated and unsupported features, and any application and server configuration updates that might be required to prevent changes in application behavior.

It also provides information about tools that can help with the migration, such as Red Hat Application Migration Toolkit, which simplifies migration of Java applications, and the JBoss Server Migration Tool, which updates the server configuration.

Once the application is successfully deployed and running, plans can be made to upgrade individual components to use the new functions and features of JBoss EAP 7.

If you plan to migrate your JBoss EAP 5 applications directly to JBoss EAP 7, see Migrating from Older Releases of JBoss EAP.

## 1.1. ABOUT MIGRATIONS AND UPGRADES

### Major Upgrades

A major upgrade or migration is required when an application is moved from one major release to another, for example, from JBoss EAP 6.4 to JBoss EAP 7.0. If an application follows the Java EE specifications, does not access deprecated APIs, and does not contain proprietary code, it might be possible to run the application in JBoss EAP 7 without any application code changes. However, the server configuration has changed in JBoss EAP 7 and requires migration. This type of migration is addressed in this guide.

### Minor Updates

JBoss EAP periodically provides point releases, which are minor updates that include bug fixes, security fixes, and new features. Information about the changes made in a point release are documented in this guide and in the *7.2.0 Release Notes*.

You can use the JBoss Server Migration Tool to automatically upgrade from one point release to another, for example from JBoss EAP 7.0 to JBoss EAP 7.1. For information about how to configure and run the tool, see Using the JBoss Server Migration Tool.

If you prefer, you can perform a manual upgrade of the server configuration. Instructions on how to perform a manual upgrade are documented in Upgrading JBoss EAP in the JBoss EAP *Patching and Upgrading Guide*.

### Cumulative Patches

JBoss EAP also periodically provides cumulative patches that contain bug and security fixes. Cumulative patches increment the release by the last digit, for example from 7.1.0 to 7.1.1. Patch installation is addressed in the JBoss EAP *Patching and Upgrading Guide*.

## 1.2. ABOUT THE USE OF EAP_HOME IN THIS DOCUMENT

In this document, the variable ***EAP_HOME*** is used to denote the path to the JBoss EAP installation. Replace this variable with the actual path to your JBoss EAP installation.

- If you installed JBoss EAP using the ZIP install method, the install directory is the **jboss-eap-7.2** directory where you extracted the ZIP archive.

- If you installed JBoss EAP using the RPM install method, the install directory is **/opt/rh/eap7/root/usr/share/wildfly/**.
- If you used the installer to install JBoss EAP, the default path for ***EAP_HOME*** is **${user.home}/EAP-7.2.0**:
    - For Red Hat Enterprise Linux and Solaris: **/home/*USER_NAME*/EAP-7.2.0/**
    - For Microsoft Windows: **C:\Users\*USER_NAME*\EAP-7.2.0\**
- If you used the Red Hat CodeReady Studio installer to install and configure the JBoss EAP server, the default path for ***EAP_HOME*** is **${user.home}/devstudio/runtimes/jboss-eap**:
    - For Red Hat Enterprise Linux: **/home/*USER_NAME*/devstudio/runtimes/jboss-eap/**
    - For Microsoft Windows: **C:\Users\*USER_NAME*\devstudio\runtimes\jboss-eap** or **C:\Documents and Settings\*USER_NAME*\devstudio\runtimes\jboss-eap\**

NOTE

***EAP_HOME*** is not an environment variable. ***JBOSS_HOME*** is the environment variable used in scripts.

# CHAPTER 2. PREPARE FOR MIGRATION

## 2.1. PREPARATION OVERVIEW

In JBoss EAP 7, an effort was made to provide backward compatibility for JBoss EAP 6 applications. However, if your application uses features that were deprecated or functionality that was removed from JBoss EAP 7, you might need to make changes to your application code.

In addition, a number of things have changed in this release that might impact deployment of JBoss EAP 7 applications. It is recommended that you do some research and planning before you attempt to migrate your application.

- Become familiar with the features of Java EE 8 .
- If you are migrating from JBoss EAP 6.4, also become familiar the features of Java EE 7 .
- Review what's new in JBoss EAP 7 .
- Review the list of deprecated and unsupported features.
- Review the material in the JBoss EAP 7 *Getting Started Guide*.
- Take a look at the tools that can help with migration tasks.

Once you are comfortable with the feature changes, the development materials, and the tools that can assist your migration efforts, you can begin to evaluate your applications and your server configuration to determine the changes that are needed to run in JBoss EAP 7.

## 2.2. REVIEW THE JAVA EE 8 FEATURES

Java EE 8 builds on Java EE 7, which included many improvements to make it easier to develop and run feature rich applications on private and public clouds. It incorporated new features and the latest standards such as HTML5, WebSocket, JSON, Batch, and Concurrency Utilities. Updates included JPA 2.1, JAX-RS 2.0, Servlet 3.1, Expression Language 3.0, JMS 2.0. JSF 2.2, EJB 3.2, CDI 1.2, and Bean Validation 1.1.

Java EE 8 enhancements include a new portable security API, support for Java Servlet 4.0 with HTTP/2 support, JPA 2.2, JAX-RS 2.1, JSF 2.3, CDI 2.0, enhanced JSON support and a new JSON binding API, support for asynchronous CDI events, and much more.

You can find more information about Java EE 8, including tutorials, on Oracle's website: Java EE at a Glance.

## 2.3. REVIEW THE JAVA EE 7 FEATURES

If you are migrating from JBoss EAP 6.4, Java EE 7 includes many improvements to make it easier to develop and run feature rich applications on private and public clouds. It incorporates new features and the latest standards such as HTML5, WebSocket, JSON, Batch, and Concurrency Utilities. Updates include JPA 2.1, JAX-RS 2.0, Servlet 3.1, Expression Language 3.0, JMS 2.0. JSF 2.2, EJB 3.2, CDI 1.2, and Bean Validation 1.1.

You can find more information about Java EE 7, including tutorials, on Oracle's web site: Java™ EE Documentation.

## 2.4. REVIEW WHAT'S NEW IN JBOSS EAP 7

JBoss EAP 7 includes some notable upgrades and improvements over previous releases. This section highlights some of the new features and enhancements that have been introduced in the JBoss EAP 7 point releases.

### New Features and Enhancements Introduced JBoss EAP 7.0

**Java EE 7**

JBoss EAP 7 is a certified implementation of Java EE 7, meeting both the Web Profile and the full platform specifications. It also includes support for the latest iterations of CDI 1.2 and Web Sockets 1.1.

**Undertow**

Undertow is the new lightweight, flexible, and performant web server included in JBoss EAP 7, replacing JBoss Web. Written in Java, it is designed for maximum throughput and scalability. It supports the latest web technologies, such as the new HTTP/2 standard.

**Apache ActiveMQ Artemis**

Apache ActiveMQ Artemis is the new JBoss EAP 7 built-in messaging provider. Based on a code donation from HornetQ, this Apache subproject provides outstanding performance based on a proven non-blocking architecture.

**IronJacamar 1.2**

The latest IronJacamar provides a stable and feature rich support for JCA and DataSources.

**JBossWS 5**

The fifth generation of JBossWS is a major leap forward, bringing new features and performance improvements to JBoss EAP 7 web services.

**RESTEasy 3**

JBoss EAP 7 includes the latest generation of RESTEasy. It goes beyond the standard Java EE REST APIs (JAX-RS 2.0) by providing a number of useful extensions such as JSON Web Encryption, Jackson, JSON-P, and Jettison.

**OpenJDK ORB**

JBoss EAP 7 replaced the JacORB IIOP implementation with a downstream branch of the OpenJDK ORB, leading to better interoperability with the JVM ORB and the Java EE RI.

**Feature Rich Clustering**

Clustering support was heavily refactored in JBoss EAP 7 and includes several public APIs for access by applications.

**Port Reduction**

By utilizing HTTP upgrade, JBoss EAP 7 has moved nearly all of its protocols to be multiplexed over just two HTTP ports: a management port (9990), and an application port (8080).

**Enhanced Logging**

The management API now supports the ability to list and view the available log files on a server, or even define custom formatters other than the default pattern formatter. Deployment's logging setup is also greatly enhanced.

For a complete list of new features introduced in JBoss EAP 7.0, see New Features and Enhancements in the JBoss EAP *7.0.0 Release Notes*.

### New Features and Enhancements Introduced in JBoss EAP 7.1

**Elytron**

Elytron, based on the WildFly Elytron project, is the new security framework in JBoss EAP 7.1. It is designed to unify security across the entire application server.

**Management Console**

The management console has been improved to provide the ability to configure more subsystems, provide enhanced **transaction** subsystem and transaction resource metrics, and manage many additional configurations.

**Management CLI**

The management CLI provides enhanced support for response and file attachments, module configuration, and debugging support through the **echo-command** argument.

For the complete list of new features introduced in JBoss EAP 7.1, see New Features and Enhancements in the *7.1.0 Release Notes* on the Red Hat Customer Portal.

### New Features and Enhancements Introduced in JBoss EAP 7.2

**Java EE 8**

JBoss EAP 7.2 is a certified implementation of Java EE 8. It includes support for Java Servlet 4.0, Java Persistence 2.2, CDI 2.0, JSF 2.3, JSON-B 1.0, JSON-P 1.1, and JAX-RS 2.1, and more. See Java™ EE 8 Technologies for more information about the technologies supported in the Java Enterprise Edition (Java EE) 8 platform.

**BOMs Available for Application Development**

A new set of BOMs are available that provide the JBoss EAP runtime dependencies for Java EE 8. Where the Java EE 7 BOM names contained **javaee7**, the BOMs in this release now contain **javaee8** in their names. For more information about the new BOMs, see Manage Project Dependencies in the *Development Guide* for JBoss EAP.

For the complete list of new features introduced in JBoss EAP 7.2, see New Features and Enhancements in the *7.2.0 Release Notes* on the Red Hat Customer Portal.

## 2.5. REVIEW THE LIST OF DEPRECATED AND UNSUPPORTED FEATURES

Before you migrate your application to JBoss EAP 7.2, be aware that some features that were available in previous releases of JBoss EAP might be deprecated or no longer supported. Support for some technologies was removed due to the high maintenance cost, low community interest, and much better alternative solutions.

The following is a short summary of some of the deprecated and unsupported features.

**EJB Entity Beans**

EJB entity beans are no longer supported. If your application uses EJB entity beans, you should migrate the code to use JPA, which offers a much more performant and flexible API.

**JAX-RPC**

Because JAX-WS offers a much more accurate and complete solution, code written for JAX-RPC should be migrated to use JAX-WS.

**JSR-88**

Java EE Application Deployment API specification (JSR-88), which defined a contract to enable tools from multiple providers to configure and deploy applications on any Java EE platform product, was not widely adopted. You must use another JBoss EAP supported option for application deployment, such as the management console, the management CLI, deployment scanner, or Maven.

**Generic JMS Resource Adapter**

The ability to configure a generic JMS resource adapter to connect to a JMS provider is no longer supported.

**IO Subsystem**

IO buffer pools are deprecated. They are replaced by Undertow byte buffer pools.

**Cache Stores**

The **remote** cache store has been deprecated in favor of using the **hotrod** cache store.

**Platforms and Features**

A number of platforms and databases that were available in previous releases are deprecated in JBoss EAP 7.2.

For a complete list of deprecated and unsupported features in JBoss EAP 7.0, see Unsupported and Deprecated Functionality in the JBoss EAP *7.0.0 Release Notes* on the Red Hat Customer Portal.

For the complete list of deprecated and unsupported features in JBoss EAP 7.1, see Unsupported and Deprecated Functionality in the JBoss EAP *7.1.0 Release Notes* on the Red Hat Customer Portal.

For the complete list of deprecated and unsupported features in JBoss EAP 7.2, see Unsupported and Deprecated Functionality in the JBoss EAP *7.2.0 Release Notes* on the Red Hat Customer Portal.

## 2.6. REVIEW THE JBOSS EAP GETTING STARTED MATERIAL

Be sure to review the JBoss EAP *Getting Started Guide*. It contains the following important information:

- How to download and install JBoss EAP 7
- How to download and install Red Hat CodeReady Studio
- How to configure Maven for your development environment, manage project dependencies, and configure your projects to use the JBoss EAP Bill of Material (BOM) artifacts
- How to download and run the quickstart example applications that ship with the product

## 2.7. MIGRATION ANALYSIS AND PLANNING

Each application and server configuration is unique, and you must thoroughly understand the components and architecture of the existing application and server platform before you attempt the migration. Your migration plan should include a detailed road map for testing and rollout to production that takes into account the following information.

**Identify the People Responsible for the Migration**

Identify the stakeholders, project managers, developers, administrators, and others who are responsible for the migration.

**Review the Application Server Platform Configuration and Hardware**

Examine the existing application server and platform configuration to determine how they are impacted by feature changes in JBoss EAP 7. The review should include the following items.

- Operating systems and versions
- Database used by the applications
- Web servers
- Security architecture

- Number and type of processors
- Amount of memory
- Amount of physical disk storage
- Migration of database or messaging data
- Other components that might be impacted by the migration

**Review the Current Production Environment**

You should plan to recreate the production environment as closely as possible for testing and staging the migration process.

- Take into account any clustering configurations. See Upgrading a Cluster in the JBoss EAP *Patching and Upgrading Guide* for more information about how to migrate clusters.
- If you are currently running a large managed domain, consider a gradual migration approach.
- Determine whether you need to migrate any database or messaging data.

**Examine and Understand the Existing Application**

Thoroughly examine the existing JBoss EAP 6 application. Be totally familiar with its architecture, functions, features and components, including:

- The JVM version
- Integration with other Red Hat application server middleware components
- Integration with proprietary third-party software
- Use of deprecated features that will require replacement
- Application configuration including deployment descriptors, JNDI, persistence, JDBC configuration and pooling, JMS topics and queues, and logging

Identify any code or configuration incompatibilities that will require modification during the migration to JBoss EAP 7.

**Create a Detailed Test Plan**

- The plan should include regression testing and acceptance criteria requirements.
- It should also include performance testing.
- Set up a staging environment as close to the production environment as possible to test the migration before the rollout to production.
- Be sure to create a backup and backout plan!

**Review the Resources Available for the Migration Process**

- Assess the skills of the development team and plan for training or additional consulting help.
- Be aware that additional hardware and other resources will be required for staging and testing during the migration process until the effort is completed.

- Determine whether any formal training is needed. If so, add it to the schedule.

**Execute the Plan**

Gather the necessary resources and implement the migration plan.

**IMPORTANT**

Before making any modifications to your application, be sure to create a backup copy.

## 2.8. BACK UP IMPORTANT DATA AND REVIEW SERVER STATE

Before you migrate your application, you need to be aware of the following potential issues.

- The migration might remove temporary folders. Any deployments stored in the **data/content/** directory must be backed up prior to the migration and restored after it completes. Otherwise, the server will fail to start due to the missing content.
- Prior to the migration, handle any open transactions and delete the **data/tx-object-store/** transaction directory.
- The persistent timer data in **data/timer-service-data** must be checked to determine whether it will still be applicable after the upgrade. Before the upgrade, review the **deployment-*** files in that directory to determine which timers are still in use.

Be sure to also back up the current server configuration and applications before you begin.

## 2.9. MIGRATING AN RPM INSTALLATION

**IMPORTANT**

It is not supported to have more than one RPM-installed instance of JBoss EAP on a single Red Hat Enterprise Linux Server. As a result, we recommend that you migrate your JBoss EAP installation to a new machine when migrating to JBoss EAP 7.

When migrating a JBoss EAP RPM installation from JBoss EAP 6 to JBoss EAP 7, ensure that JBoss EAP 7 is installed on a machine that does not have an existing JBoss EAP RPM installation.

To install JBoss EAP 7 using RPMs, see the JBoss EAP *Installation Guide*.

The migration advice in this guide also applies to migrating RPM installations of JBoss EAP, but you might need to alter some steps (such as how to start JBoss EAP) to suit an RPM installation compared to a ZIP or installer installation.

## 2.10. MIGRATE JBOSS EAP RUNNING AS A SERVICE

If you run JBoss EAP 6 as a service, be sure to review Configuring JBoss EAP to Run as a Service in the JBoss EAP *Installation Guide* for updated configuration instructions for JBoss EAP 7.

## 2.11. MIGRATE TO OPENSHIFT JDK 11 IMAGE

To migrate from the JBoss EAP for OpenShift JDK 8 image to OpenShift JDK 11 image, see Migrating to OpenShift JDK 11 Image in the Getting Started with JBoss EAP for OpenShift Container Platform Guide.

# CHAPTER 3. TOOLS TO ASSIST IN MIGRATION

## 3.1. USE RED HAT APPLICATION MIGRATION TOOLKIT TO ANALYZE APPLICATIONS FOR MIGRATION

Red Hat Application Migration Toolkit (RHAMT) is an extensible and customizable rule-based set of tools that helps simplify migration of Java applications. It analyzes the APIs, technologies, and architectures used by the applications you plan to migrate and provides detailed migration reports for each application. These reports provide the following information.

- Detailed explanations of the migration changes needed
- Whether the reported change is mandatory or optional
- Whether the reported change is complex or trivial
- Links to the code requiring the migration change
- Hints and links to information about how to make the required changes
- An estimate of the level of effort for each migration issue found and the total estimated effort to migrate the application

You can use RHAMT to analyze the code and architecture of your JBoss EAP 6 applications before you migrate them to JBoss EAP 7. The RHAMT rule set for migration from JBoss EAP 6 to JBoss EAP 7 reports on XML descriptors and specific application code and parameters that need to be replaced by an alternative configuration when migrating to JBoss EAP 7.

For more information about how to use Red Hat Application Migration Toolkit to analyze your JBoss EAP 6 applications, see the Red Hat Application Migration Toolkit *Getting Started Guide*.

## 3.2. USE THE JBOSS SERVER MIGRATION TOOL TO MIGRATE SERVER CONFIGURATIONS

The JBoss Server Migration Tool is the preferred method to update your server configuration to include the new features and settings in JBoss EAP 7 while keeping your existing configuration. The JBoss Server Migration Tool reads your existing JBoss EAP server configuration files and adds configurations for any new subsystems, updates the existing subsystem configurations with new features, and removes any obsolete subsystem configurations.

You can use the JBoss Server Migration Tool to migrate standalone servers and managed domains for the following configurations.

**Migrating to JBoss EAP 7.2**

The JBoss Server Migration Tool ships with JBoss EAP 7.2, so there is no separate download or installation required. This tool supports migration from JBoss EAP 6.4 and later to JBoss EAP 7.2. You run the tool by executing the **jboss-server-migration** script located in the ***EAP_HOME*/bin** directory. For more information about how to configure and run the tool, see Using the JBoss Server Migration Tool.
It is recommended that you use this version of the JBoss Server Migration Tool to migrate your server configuration to JBoss EAP 7.2 as this version of the tool is supported.

**Migrating from WildFly to JBoss EAP**

If you want to migrate from the WildFly server to JBoss EAP, you must download the latest binary distribution of the JBoss Server Migration Tool from the JBoss Server Migration Tool GitHub repository. This open source, standalone version of the tool supports migration from several versions of the WildFly server to JBoss EAP. For information about how to install and run this version of the tool, see the JBoss Server Migration Tool *User Guide*.

**IMPORTANT**

The binary distribution of the JBoss Server Migration Tool is not supported. If you are migrating from a previous release of JBoss EAP, it is recommended that you use this supported version of the tool to migrate your server configuration to JBoss EAP 7.2 instead.

# CHAPTER 4. SERVER CONFIGURATION CHANGES

## 4.1. RPM INSTALLATION CHANGES

In JBoss EAP 6, the default path for the RPM installation was the **/usr/share/jbossas/** directory.

JBoss EAP 7 was built to Software Collections Library conventions. The root directory of Software Collections is normally located in the **/opt/** directory to avoid possible conflicts between Software Collections and the base system installation. The use of the **/opt/** directory is recommended by the Filesystem Hierarchy Standard (FHS). As a result, the default path for the RPM installation has changed to **/opt/rh/eap7/root/usr/share/wildfly/** in JBoss EAP 7.

## 4.2. SERVER CONFIGURATION MIGRATION OPTIONS

To migrate your server configuration from JBoss EAP 6 to JBoss EAP 7, you can either use the JBoss Server Migration Tool or you can perform a manual migration with the help of the management CLI **migrate** operation.

### JBoss Server Migration Tool

The JBoss Server Migration Tool is the preferred method to update your configuration to include the new features and settings in JBoss EAP 7 while keeping your existing configuration. For information about how to configure and run the tool, see Using the JBoss Server Migration Tool.

### Management CLI Migrate Operation

You can use the management CLI **migrate** operation to update the **jacorb**, **messaging**, and **web** subsystems in the JBoss EAP 6 server configuration file to allow them run on the new release, but be aware that the result is not a complete JBoss EAP 7 configuration. For example:

- The operation does not update the original **remote** protocol and port settings to the new **http-remoting** and new port settings now used in JBoss EAP 7.
- The configuration does not include the new JBoss EAP subsystems or features such as clustered singleton deployments, or graceful shutdown.
- The configuration does not include the new Java EE 7 features such as batch processing.
- The **migrate** operation does not migrate the **ejb3** subsystem configuration. For information about possible EJB migration issues, see EJB Server Configuration Changes.

For more information about using the **migrate** operation to migration the server configuration, see Management CLI Migration Operation.

## 4.3. MANAGEMENT CLI MIGRATION OPERATION

You can use the management CLI to update your JBoss EAP 6 server configuration files to run on JBoss EAP 7. The management CLI provides a **migrate** operation to automatically update the **jacorb**, **messaging**, and **web** subsystems from the previous release to the new configuration. You can also execute the **describe-migration** operation for the **jacorb**, **messaging**, and **web** subsystems to review the proposed migration configuration changes before you perform the migration. There are no replacements for the **cmp**, **jaxr**, or **threads** subsystems and they must be removed from the server configuration.

IMPORTANT

See Server Configuration Migration Options for limitations of the **migrate** operation. The JBoss Server Migration Tool is the preferred method to update your configuration to include the new features and settings in JBoss EAP 7 while keeping your existing configuration. For information about how to configure and run the tool, see Using the JBoss Server Migration Tool.

Table 4.1. Subsystem Migration and Management CLI Operation

| JBoss EAP 6 Subsystem | JBoss EAP 7 Subsystem | Management CLI Operation |
|---|---|---|
| cmp | no replacement | remove |
| jacorb | iiop-openjdk | migrate |
| jaxr | no replacement | remove |
| messaging | messaging-activemq | migrate |
| threads | no replacement | remove |
| web | undertow | migrate |

### Start the Server and the Management CLI

Follow the steps below to update your JBoss EAP 6 server configuration to run on JBoss EAP 7.

1. Before you begin, review Back Up Important Data and Review Server State . It contains important information about making sure the server is in a good state and the appropriate files are backed up.

2. Start the JBoss EAP 7 server with the JBoss EAP 6 configuration.

   a. Back up the JBoss EAP 7 server configuration files.

   b. Copy the configuration file from the previous release into the JBoss EAP 7 directory.

   ```
   $ cp EAP6_HOME/standalone/configuration/standalone-full.xml
   EAP7_HOME/standalone/configuration
   ```

   c. Navigate to the JBoss EAP 7 install directory and start the server with the **--start-mode=admin-only** argument.

   ```
   $ bin/standalone.sh -c standalone-full.xml --start-mode=admin-only
   ```

NOTE

You will see the following **org.jboss.as.controller.management-operation** ERRORS in the server log when you start the server. These errors are expected and indicate that the legacy subsystem configurations must be removed or migrated to JBoss EAP 7.

- WFLYCTL0402: Subsystems [cmp] provided by legacy extension 'org.jboss.as.cmp' are not supported on servers running this version. Both the subsystem and the extension must be removed or migrated before the server will function.
- WFLYCTL0402: Subsystems [jacorb] provided by legacy extension 'org.jboss.as.jacorb' are not supported on servers running this version. Both the subsystem and the extension must be removed or migrated before the server will function.
- WFLYCTL0402: Subsystems [jaxr] provided by legacy extension 'org.jboss.as.jaxr' are not supported on servers running this version. Both the subsystem and the extension must be removed or migrated before the server will function.
- WFLYCTL0402: Subsystems [messaging] provided by legacy extension 'org.jboss.as.messaging' are not supported on servers running this version. Both the subsystem and the extension must be removed or migrated before the server will function.
- WFLYCTL0402: Subsystems [threads] provided by legacy extension 'org.jboss.as.threads' are not supported on servers running this version. Both the subsystem and the extension must be removed or migrated before the server will function.
- WFLYCTL0402: Subsystems [web] provided by legacy extension 'org.jboss.as.web' are not supported on servers running this version. Both the subsystem and the extension must be removed or migrated before the server will function.

3. Open a new terminal, navigate to the JBoss EAP 7 install directory, and start the management CLI using the **--controller=remote://localhost:9990** arguments.

```
$ bin/jboss-cli.sh --connect --controller=remote://localhost:9990
```

## Migrate the JacORB, Messaging, and Web Subsystems

1. To review the configuration changes that will be made to the subsystem before you perform the migration, execute the **describe-migration** operation.
   The **describe-migration** operation uses the following syntax.

```
/subsystem=SUBSYSTEM_NAME:describe-migration
```

The following example describes the configuration changes that are made to the JBoss EAP 6.4 **standalone-full.xml** configuration file when it is migrated to JBoss EAP 7. Entries were removed from the output to improve readability and to save space.

### Example: describe-migration Operation

_

```
/subsystem=messaging:describe-migration
{
    "outcome" => "success",
    "result" => {
        "migration-warnings" => [],
        "migration-operations" => [
            {
                "operation" => "add",
                "address" => [("extension" => "org.wildfly.extension.messaging-activemq")],
                "module" => "org.wildfly.extension.messaging-activemq"
            },
            {
                "operation" => "add",
                "address" => [("subsystem" => "messaging-activemq")]
            },
            <!-- *** Entries removed for readability *** -->
            {
                "operation" => "remove",
                "address" => [("subsystem" => "messaging")]
            },
            {
                "operation" => "remove",
                "address" => [("extension" => "org.jboss.as.messaging")]
            }
        ]
    }
}
```

2. Execute the **migrate** operation to migrate the subsystem configuration to the subsystem that replaces it in JBoss EAP 7. The operation uses the following syntax.

```
/subsystem=SUBSYSTEM_NAME:migrate
```

NOTE

The **messaging** subsystem **describe-migration** and **migrate** operations allow you to pass an argument to configure access by legacy clients. For more information about the command syntax, see Messaging Subsystem Migration and Forward Compatibility.

3. Review the outcome and result of the command. Be sure the operation completed successfully and there are no "migration-warning" entries. This means the migration configuration for the subsystem is complete.

**Example: Successful migrate Operation with No Warnings**

```
/subsystem=messaging:migrate
{
    "outcome" => "success",
    "result" => {"migration-warnings" => []}
}
```

If you see "migration-warnings" entries in the log, this indicates the migration of the server configuration completed successfully but it was not able to migrate all of elements and attributes. You must follow the suggestions provided by the "migration-warnings" and run additional management CLI commands to modify those configurations. The following is an example of a **migrate** operation that returns "migration-warnings".

### Example: migrate Operation with Warnings

```
/subsystem=messaging:migrate
{
    "outcome" => "success",
    "result" => {"migration-warnings" => [
        "WFLYMSG0080: Could not migrate attribute group-address from resource [
    (\"subsystem\" => \"messaging-activemq\"),
    (\"server\" => \"default\"),
    (\"broadcast-group\" => \"groupB\")
]. Use instead the socket-binding attribute to configure this broadcast-group.",
        "WFLYMSG0080: Could not migrate attribute group-port from resource [
    (\"subsystem\" => \"messaging-activemq\"),
    (\"server\" => \"default\"),
    (\"broadcast-group\" => \"groupB\")
]. Use instead the socket-binding attribute to configure this broadcast-group.",
        "WFLYMSG0080: Could not migrate attribute local-bind-address from resource [
    (\"subsystem\" => \"messaging-activemq\"),
    (\"server\" => \"default\"),
    (\"broadcast-group\" => \"groupA\")
]. Use instead the socket-binding attribute to configure this broadcast-group.",
        "WFLYMSG0080: Could not migrate attribute local-bind-port from resource [
    (\"subsystem\" => \"messaging-activemq\"),
    (\"server\" => \"default\"),
    (\"broadcast-group\" => \"groupA\")
]. Use instead the socket-binding attribute to configure this broadcast-group.",
        "WFLYMSG0080: Could not migrate attribute group-address from resource [
    (\"subsystem\" => \"messaging-activemq\"),
    (\"server\" => \"default\"),
    (\"broadcast-group\" => \"groupA\")
]. Use instead the socket-binding attribute to configure this broadcast-group.",
        "WFLYMSG0080: Could not migrate attribute group-port from resource [
    (\"subsystem\" => \"messaging-activemq\"),
    (\"server\" => \"default\"),
    (\"broadcast-group\" => \"groupA\")
]. Use instead the socket-binding attribute to configure this broadcast-group."
    ]}
}
```

#### NOTE

The list of **migrate** and **describe-migration** warnings for each subsystem is located in the Reference Material at the end of this guide.

- Jacorb Subsystem Migration Operation Warnings
- Messaging Subsystem Migration Operation Warnings
- Web Subsystem Migration Operation Warnings

4. Review the server configuration file to verify the extension, subsystem, and namespace were updated and the existing subsystem configuration was migrated to JBoss EAP 7.

   NOTE

   You must repeat this process for each of the **jacorb**, **messaging**, and **web** subsystems using the following commands.

   ```
   /subsystem=jacorb:migrate
   /subsystem=messaging:migrate
   /subsystem=web:migrate
   ```

5. Remove the **cmp**, **jaxr**, and **threads** subsystems and extensions from the server configuration. While still in the management CLI prompt, remove the obsolete **cmp**, **jaxr**, and **threads** subsystems by executing the following commands.

   ```
   /subsystem=cmp:remove
   /extension=org.jboss.as.cmp:remove
   /subsystem=jaxr:remove
   /extension=org.jboss.as.jaxr:remove
   /subsystem=threads:remove
   /extension=org.jboss.as.threads:remove
   ```

IMPORTANT

You must migrate the **messaging**, **jacorb**, and **web** subsystems and remove the **cmp**, **jaxr**, and **threads** extensions and subsystems before you can restart the server for normal operation. If you need to restart the server before you complete this process, be sure to include the **--start-mode=admin-only** argument on the server start command line. This allows you to continue with the configuration changes.

## 4.4. LOGGING CHANGES

### 4.4.1. Logging Message Prefix Changes

Log messages are prefixed with the project code for the subsystem that reports the message. The prefixes for all log messages have changed in JBoss EAP 7.

For a complete list of the new log message project code prefixes used in JBoss EAP 7, see Project Codes Used in JBoss EAP in the JBoss EAP *Development Guide*.

### 4.4.2. Root Logger Console Handler Changes

The JBoss EAP 7.0 root logger included a console log handler for all domain server profiles and for all default standalone profiles except the **standalone-full-ha** profile. As of JBoss EAP 7.1, the root logger no longer includes a console log handler for the managed domain profiles. The host controller and process controller log to the console by default. To achieve the same functionality that was provided in JBoss EAP 7.0, see Configure a Console Log Handler in the *Configuration Guide* for JBoss EAP.

## 4.5. WEB SERVER CONFIGURATION CHANGES

### 4.5.1. Replace the Web Subsystem with Undertow

Undertow replaces JBoss Web as the web server in JBoss EAP 7. This means the legacy **web** subsystem configuration must be migrated to the new JBoss EAP 7 **undertow** subsystem configuration.

- The **urn:jboss:domain:web:2.2** subsystem configuration namespace in the server configuration file has been replaced by the **urn:jboss:domain:undertow:7.0** namespace.
- The **org.jboss.as.web** extension module, located in ***EAP_HOME*/modules/system/layers/base/**, has been replaced with the **org.wildfly.extension.undertow** extension module.

You can use the management CLI **migrate** operation to migrate the **web** subsystem to **undertow** in the server configuration file. However, be aware that this operation is not able to migrate all JBoss Web subsystem configurations. If you see "migration-warning" entries, you must run additional management CLI commands to migrate those configurations to Undertow. For more information about the management CLI **migrate** operation, see Management CLI Migration Operation.

The following is an example of the default **web** subsystem configuration in JBoss EAP 6.4.

```
<subsystem xmlns="urn:jboss:domain:web:2.2" default-virtual-server="default-host" native="false">
    <connector name="http" protocol="HTTP/1.1" scheme="http" socket-binding="http"/>
    <virtual-server name="default-host" enable-welcome-root="true">
        <alias name="localhost"/>
        <alias name="example.com"/>
    </virtual-server>
</subsystem>
```

The following is an example of the default **undertow** subsystem configuration in JBoss EAP 7.2.

```
<subsystem xmlns="urn:jboss:domain:undertow:7.0" default-server="default-server" default-virtual-
host="default-host" default-servlet-container="default" default-security-domain="other">
    <buffer-cache name="default"/>
    <server name="default-server">
        <http-listener name="default" socket-binding="http" redirect-socket="https" enable-http2="true"/>
        <https-listener name="https" socket-binding="https" security-realm="ApplicationRealm" enable-
http2="true"/>
        <host name="default-host" alias="localhost">
            <location name="/" handler="welcome-content"/>
            <http-invoker security-realm="ApplicationRealm"/>
        </host>
    </server>
    ...
</subsystem>
```

### 4.5.2. Migrate JBoss Web Rewrite Conditions

The management CLI **migrate** operation is not able to automatically migrate rewrite conditions. They are reported as "migration-warnings", and you must migrate them manually. You can create the equivalent configuration in JBoss EAP 7 by using Undertow Predicates Attributes and Handlers .

The following is an example of a **web** subsystem configuration in JBoss EAP 6 that includes **rewrite** configuration.

```
<subsystem xmlns="urn:jboss:domain:web:2.2" default-virtual-server="default" native="false">
```

```
    <virtual-server name="default" enable-welcome-root="true">
        <alias name="localhost"/>
        <rewrite name="test" pattern="(.*)/toberewritten/(.*)" substitution="$1/rewritten/$2" flags="NC"/>
        <rewrite name="test2" pattern="(.*)" substitution="-" flags="F">
            <condition name="get" test="%{REQUEST_METHOD}" pattern="GET"/>
            <condition name="andCond" test="%{REQUEST_URI}" pattern=".*index.html" flags="NC"/>
        </rewrite>
    </virtual-server>
</subsystem>
```

Follow the Management CLI Migration Operation instructions to start your server and the management CLI, then migrate the **web** subsystem configuration file using the following command.

```
/subsystem=web:migrate
```

The following "migration-warnings" are reported when you run the **migrate** operation on the above configuration.

```
/subsystem=web:migrate
{
    "outcome" => "success",
    "result" => {"migration-warnings" => [
        "WFLYWEB0002: Could not migrate resource {
    \"pattern\" => \"(.*)\",
    \"substitution\" => \"-\",
    \"flags\" => \"F\",
    \"operation\" => \"add\",
    \"address\" => [
        (\"subsystem\" => \"web\"),
        (\"virtual-server\" => \"default-host\"),
        (\"rewrite\" => \"test2\")
    ]
}",
        "WFLYWEB0002: Could not migrate resource {
    \"test\" => \"%{REQUEST_METHOD}\",
    \"pattern\" => \"GET\",
    \"flags\" => undefined,
    \"operation\" => \"add\",
    \"address\" => [
        (\"subsystem\" => \"web\"),
        (\"virtual-server\" => \"default-host\"),
        (\"rewrite\" => \"test2\"),
        (\"condition\" => \"get\")
    ]
}",
        "WFLYWEB0002: Could not migrate resource {
    \"test\" => \"%{REQUEST_URI}\",
    \"pattern\" => \".*index.html\",
    \"flags\" => \"NC\",
    \"operation\" => \"add\",
    \"address\" => [
        (\"subsystem\" => \"web\"),
        (\"virtual-server\" => \"default-host\"),
        (\"rewrite\" => \"test2\"),
        (\"condition\" => \"andCond\")
    ]
}"
    ]}
}
```

Review the server configuration file and you see the following configuration for the **undertow** subsystem.

NOTE

The rewrite configuration is dropped.

```
<subsystem xmlns="urn:jboss:domain:undertow:7.0" default-server="default-server" default-virtual-host="default-host" default-servlet-container="default" default-security-domain="other">
    <buffer-cache name="default"/>
    <server name="default-server">
        <http-listener name="http" socket-binding="http"/>
        <https-listener name="https" socket-binding="https" security-realm="ApplicationRealm" enable-http2="true"/>
        <host name="default-host" alias="localhost, example.com">
            <location name="/" handler="welcome-content"/>
        </host>
    </server>
    <servlet-container name="default">
        <jsp-config/>
    </servlet-container>
    <handlers>
        <file name="welcome-content" path="${jboss.home.dir}/welcome-content"/>
    </handlers>
 </subsystem>
```

Use the management CLI to create the filter to replace the rewrite configuration in the **undertow** subsystem. You should see "{"outcome" ⇒ "success"}" for each command.

```
# Create the filters
/subsystem=undertow/configuration=filter/expression-filter="test1":add(expression="path('(.*)/toberewritten/(.*)') -> rewrite('$1/rewritten/$2')")
/subsystem=undertow/configuration=filter/expression-filter="test2":add(expression="method('GET') and path('.*index.html') -> response-code(403)")

# Add the filters to the default server
/subsystem=undertow/server=default-server/host=default-host/filter-ref="test1":add
/subsystem=undertow/server=default-server/host=default-host/filter-ref="test2":add
```

Review the updated server configuration file. The JBoss Web subsystem is now completely migrated and configured in the **undertow** subsystem.

```
<subsystem xmlns="urn:jboss:domain:undertow:7.0" default-server="default-server" default-virtual-host="default-host" default-servlet-container="default" default-security-domain="other">
    <buffer-cache name="default"/>
    <server name="default-server">
        <http-listener name="http" socket-binding="http"/>
        <https-listener name="https" socket-binding="https" security-realm="ApplicationRealm" enable-http2="true"/>
        <host name="default-host" alias="localhost, example.com">
            <location name="/" handler="welcome-content"/>
            <filter-ref name="test1"/>
            <filter-ref name="test2"/>
        </host>
    </server>
    <servlet-container name="default">
        <jsp-config/>
    </servlet-container>
    <handlers>
        <file name="welcome-content" path="${jboss.home.dir}/welcome-content"/>
    </handlers>
    <filters>
        <expression-filter name="test1" expression="path('(.*)/toberewritten/(.*)') ->
rewrite('$1/rewritten/$2')"/>
        <expression-filter name="test2" expression="method('GET') and path('.*index.html') -> response-
code(403)"/>
    </filters>
</subsystem>
```

For more information about how to configure filters and handlers using the management CLI, see Configuring the Web Server in the JBoss EAP 7 *Configuration Guide*.

### 4.5.3. Migrate JBoss Web System Properties

In the previous release of JBoss EAP, system properties could be used to modify the default JBoss Web behavior. For information about how to configure the same behavior in Undertow, see JBoss Web System Properties Migration Reference

### 4.5.4. Update the Access Log Header Pattern

When you migrate from JBoss EAP 6.4 to JBoss EAP 7, you might find that the access logs no longer write the expected "Referer" and "User-agent" values. This is because JBoss Web, which was included in JBoss EAP 6.4, used a pattern of **%{headername}i** in the **access-log** to log an incoming header.

**Example: Access Log Format in JBoss EAP 6.4**

```
<access-log pattern="%h %l %u %t &quot;%T sec&quot; &quot;%r&quot; %s %b &quot;%
{Referer}i&quot; &quot;%{User-agent}i&quot;"/>
```

With the change to use Undertow in JBoss EAP 7, the pattern for an incoming header has changed to **%{i,headername}**.

**Example: Access Format Header in JBoss EAP 7**

```
<access-log pattern="%h %l %u %t &quot;%T sec&quot; &quot;%r&quot; %s %b &quot;%
{i,Referer}&quot; &quot;%{i,User-Agent}&quot;"/>
```

### 4.5.5. Migrate Global Valves

Previous releases of JBoss EAP supported valves. Valves are custom classes inserted into the request processing pipeline for an application before servlet filters to make changes to the request or perform additional processing.

- Global valves are inserted into the request processing pipeline of all deployed applications and are configured in the server configuration file.
- Authenticator valves authenticate the credentials of the request.
- Custom application valves were created by extending the **org.apache.catalina.valves.ValveBase** class and configured in the **<valve>** element of the **jboss-web.xml** descriptor file. These valves must be migrated manually.

This section describes how to migrate global valves. Migration of custom and authenticator valves are covered in the Migrate Custom Application Valves section of this guide.

Undertow, which replaces JBoss Web in JBoss EAP 7, does not support global valves; however, you should be able to achieve similar functionality by using Undertow handlers. Undertow includes a number of built-in handlers that provide common functionality. It also provides the ability to create custom handlers, which can be used to replace custom valve functionality.

If your application uses valves, you must replace them with the appropriate Undertow handler code to achieve the same functionality when you migrate to JBoss EAP 7.

For more information about how to configure handlers, see Configuring Handlers in the JBoss EAP 7 *Configuration Guide*.

For more information about how to configure filters, see Configuring Filters in the JBoss EAP 7 *Configuration Guide*.

**Migrate JBoss Web Valves**
The following table lists the valves that were provided by JBoss Web in the previous release of JBoss EAP and the corresponding Undertow built-in handler. The JBoss Web valves are located in the **org.apache.catalina.valves** package.

**Table 4.2. Mapping Valves to Handlers**

| Valve | Handler |
|---|---|
| AccessLogValve | io.undertow.server.handlers.accesslog.AccessLogHandler |
| CrawlerSessionManagerValve | io.undertow.servlet.handlers.CrawlerSessionManagerHandler |
| ExtendedAccessLogValve | io.undertow.server.handlers.accesslog.AccessLogHandler |
| JDBCAccessLogValve | See the **JDBCAccessLogValve** Manual Migration Procedure below for instructions. |
| RemoteAddrValve | io.undertow.server.handlers.IPAddressAccessControlHandler |
| RemoteHostValve | io.undertow.server.handlers.AccessControlListHandler |
| RemoteIpValve | io.undertow.server.handlers.ProxyPeerAddressHandler |
| RequestDumperValve | io.undertow.server.handlers.RequestDumpingHandler |

| Valve | Handler |
|---|---|
| RewriteValve | See Migrate JBoss Web Rewrite Conditions for instructions to migrate these valves manually. |
| StuckThreadDetectionValve | io.undertow.server.handlers.StuckThreadDetectionHandler |

You can use the management CLI **migrate** operation to automatically migrate global valves that meet the following criteria:

- They are limited to the valves listed in the previous table that do not require manual processing.
- They must be defined in the **web** subsystem of the server configuration file.

For more information about the management CLI **migrate** operation, see Management CLI Migration Operation.

**JDBCAccessLogValve Manual Migration Procedure**

The **org.apache.catalina.valves.JDBCAccessLogValve** valve is an exception to the rule and can not be automatically migrated to **io.undertow.server.handlers.JDBCLogHandler**. Follow the steps below to migrate the following example valve.

```
<valve name="jdbc" module="org.jboss.as.web" class-
name="org.apache.catalina.valves.JDBCAccessLogValve">
    <param param-name="driverName" param-value="com.mysql.jdbc.Driver" />
    <param param-name="connectionName" param-value="root" />
    <param param-name="connectionPassword" param-value="password" />
    <param param-name="connectionURL" param-value="jdbc:mysql://localhost:3306/wildfly?
zeroDateTimeBehavior=convertToNull" />
    <param param-name="format" param-value="combined" />
</valve>
```

1. Create a driver module for the database that will store the log entries.
2. Configure the datasource for the database and add the driver to the list of available drivers in the **datasources** subsystem.

```
<datasources>
    <datasource jndi-name="java:jboss/datasources/accessLogDS" pool-
name="accessLogDS" enabled="true" use-java-context="true">
        <connection-url>jdbc:mysql://localhost:3306/wildfly?
zeroDateTimeBehavior=convertToNull</connection-url>
        <driver>mysql</driver>
        <security>
           <user-name>root</user-name>
           <password>Password1!</password>
        </security>
    </datasource>
    ...
    <drivers>
        <driver name="mysql" module="com.mysql">
            <driver-class>com.mysql.jdbc.Driver</driver-class>
        </driver>
      ...
    </drivers>
</datasources>
```

3. Configure an **expression-filter** in the **undertow** subsystem with the following expression: **jdbc-access-log(datasource=*DATASOURCE_JNDI_NAME*)**.

```
<filters>
    <expression-filter name="jdbc-access" expression="jdbc-access-
log(datasource='java:jboss/datasources/accessLogDS')" />
    ...
</filters>
```

### 4.5.6. Changes to Set-Cookie Behavior

Previous specifications for **Set-Cookie** HTTP response header syntax, for example RFC2109 and RFC2965, allowed white space and other separator characters in the cookie value when the cookie value was quoted. JBoss Web in JBoss EAP 6.4 conformed to the previous specifications and automatically quoted a cookie value when it contained any separator characters.

The RFC6265 specification for **Set-Cookie** HTTP response header syntax states that cookie values in the **Set-Cookie** response header must conform to specific grammar constraints. For example, they must be US-ASCII characters, but they cannot include CTRLs (controls), whitespace, double quotes, commas, semicolons, or backslash characters.

In JBoss EAP 7.0, prior to cumulative patch Red Hat JBoss Enterprise Application Platform 7.0 Update 08, Undertow does not restrict these invalid characters and does not quote cookies that contained the excluded characters. If you apply this cumulative patch or a newer cumulative patch you can enable RFC6265 compliant cookie validation by setting the **io.undertow.cookie.DEFAULT_ENABLE_RFC6265_COOKIE_VALIDATION** system property to **true**.

Starting in JBoss EAP 7.1, by default, Undertow does not enable RFC6265 compliant cookie validation. It does quote cookies that contain the excluded characters. Starting in JBoss EAP 7.1, you cannot use the **io.undertow.cookie.DEFAULT_ENABLE_RFC6265_COOKIE_VALIDATION** system property to enable RFC6265 compliant cookie validation. Instead, you enable RFC6265 compliant cookie validation for an HTTP, HTTPS, or AJP listener by setting the **rfc6265-cookie-validation** listener attribute to **true**. The default value for this attribute is **false**. The following example enables RFC6265 compliant cookie validation for the HTTP listener.

```
/subsystem=undertow/server=default-server/http-listener=default:write-attribute(name=rfc6265-
cookie-validation,value=true)
```

### 4.5.7. Changes to HTTP Method Call Behavior

JBoss EAP 6.4, which included JBoss Web as the web server, allowed HTTP **TRACE** method calls by default.

Undertow, which replaces JBoss Web as the web server in JBoss EAP 7, disallows HTTP **TRACE** method calls by default. This setting is configured using the **disallowed-methods** attribute of the **http-listener** element in the **undertow** subsystem. This can be confirmed by reviewing the output from the following **read-resource** command. Note that the value for the **disallowed-methods** attribute is **["TRACE"]**.

```
/subsystem=undertow/server=default-server/http-listener=default:read-resource
```

```
{
    "outcome" => "success",
    "result" => {
        "allow-encoded-slash" => false,
        "allow-equals-in-cookie-value" => false,
        "allow-unescaped-characters-in-url" => false,
        "always-set-keep-alive" => true,
        "buffer-pipelined-data" => false,
        "buffer-pool" => "default",
        "certificate-forwarding" => false,
        "decode-url" => true,
        "disallowed-methods" => ["TRACE"],
         ...
    }
}
```

To enable HTTP **TRACE** method calls in JBoss EAP 7 and later, you must remove the "TRACE" entry from the **disallowed-methods** attribute list by running the following command.

```
/subsystem=undertow/server=default-server/http-listener=default:list-remove(name=disallowed-methods,value="TRACE")
```

When you run the **read-resource** command again, you will notice the **TRACE** method call is no longer in the list of disallowed methods.

```
/subsystem=undertow/server=default-server/http-listener=default:read-resource
{
    "outcome" => "success",
    "result" => {
        "allow-encoded-slash" => false,
        "allow-equals-in-cookie-value" => false,
        "allow-unescaped-characters-in-url" => false,
        "always-set-keep-alive" => true,
        "buffer-pipelined-data" => false,
        "buffer-pool" => "default",
        "certificate-forwarding" => false,
        "decode-url" => true,
        "disallowed-methods" => [],
         ...
    }
}
```

For more information about the default behavior of HTTP methods, see Default Behavior of HTTP Methods in the JBoss EAP *Configuration Guide*.

### 4.5.8. Changes in the Default Web Module Behavior

In JBoss EAP 7.0, the root context of a web application was disabled by default in mod_cluster.

As of JBoss EAP 7.1, this is no longer the case. This can have unexpected consequences if you are expecting the root context to be disabled. For example, requests can be misrouted to undesired nodes or a private application that should not be exposed can be inadvertently accessible through a public proxy. Undertow locations are also now registered with the mod_cluster load balancer automatically unless they are explicitly excluded.

Use the following management CLI command to exclude ROOT from the **modcluster** subsystem configuration.

```
/subsystem=modcluster/mod-cluster-config=configuration:write-attribute(name=excluded-contexts,value=ROOT)
```

Use the following management CLI command to disable the default welcome web application.

```
/subsystem=undertow/server=default-server/host=default-host/location=\/:remove
/subsystem=undertow/configuration=handler/file=welcome-content:remove
reload
```

For more information about how to configure the default welcome web application, see Configure the Default Welcome Web Application in the *Development Guide* for JBoss EAP.

### 4.5.9. Changes in the Undertow Subsystem Default Configuration

Prior to JBoss EAP 7.2, the default **undertow** subsystem configuration included two response header filters that were appended to each HTTP response by the **default-host**.

- **Server**, which was set to **JBoss-EAP/7**.
- **X-Powered-By**, which was set to **Undertow/1**.

These response header filters were removed from the default JBoss EAP 7.2 configuration to prevent inadvertent disclosure of information about the server in use.

The following is an example of the default **undertow** subsystem configuration in JBoss EAP 7.1.

```
<subsystem xmlns="urn:jboss:domain:undertow:4.0">
    <buffer-cache name="default"/>
    <server name="default-server">
        <http-listener name="default" socket-binding="http" redirect-socket="https"/>
        <https-listener name="https" socket-binding="https" security-realm="ApplicationRealm" enable-http2="true"/>
        <host name="default-host" alias="localhost">
            <location name="/" handler="welcome-content"/>
            <filter-ref name="server-header"/>
            <filter-ref name="x-powered-by-header"/>
            <http-invoker security-realm="ApplicationRealm"/>
        </host>
    </server>
    <servlet-container name="default">
        <jsp-config/>
        <websockets/>
    </servlet-container>
    <handlers>
        <file name="welcome-content" path="${jboss.home.dir}/welcome-content"/>
    </handlers>
    <filters>
        <response-header name="server-header" header-name="Server" header-value="JBoss-EAP/7"/>
        <response-header name="x-powered-by-header" header-name="X-Powered-By" header-value="Undertow/1"/>
    </filters>
</subsystem>
```

The following is an example of the new default **undertow** subsystem configuration in JBoss EAP 7.2.

```
<subsystem xmlns="urn:jboss:domain:undertow:7.0" default-server="default-server" default-virtual-
host="default-host" default-servlet-container="default" default-security-domain="other">
    <buffer-cache name="default"/>
    <server name="default-server">
        <http-listener name="default" socket-binding="http" redirect-socket="https" enable-http2="true"/>
        <https-listener name="https" socket-binding="https" security-realm="ApplicationRealm" enable-
http2="true"/>
        <host name="default-host" alias="localhost">
            <location name="/" handler="welcome-content"/>
            <http-invoker security-realm="ApplicationRealm"/>
        </host>
    </server>
    <servlet-container name="default">
        <jsp-config/>
        <websockets/>
    </servlet-container>
    <handlers>
        <file name="welcome-content" path="${jboss.home.dir}/welcome-content"/>
    </handlers>
</subsystem>
```

## 4.6. JGROUPS SERVER CONFIGURATION CHANGES

### 4.6.1. JGroups Defaults to a Private Network Interface

In the JBoss EAP 6 default configuration, JGroups used the **public** interface defined in the **<interfaces>** section of the server configuration file.

Because it is a recommended practice to use a dedicated network interface, JGroups now defaults to using the new **private** interface that is defined in the **<interfaces>** section of the server configuration file in JBoss EAP 7.

### 4.6.2. JGroups Channels Changes

JGroups provides group communication support for HA services in the form of JGroups channels. JBoss EAP 7 introduces **<channel>** elements to the **jgroups** subsystem in the server configuration file. You can add, remove, or change the configuration of JGroups channels using the management CLI.

For more information about how to configure JGroups, see Cluster Communication with JGroups in the JBoss EAP *Configuration Guide*.

## 4.7. INFINISPAN SERVER CONFIGURATION CHANGES

### 4.7.1. Infinispan Default Cache Configuration Changes

In JBoss EAP 6, the default clustered caches for web session replication and EJB replication were replicated **ASYNC** caches. This has changed in JBoss EAP 7. The default clustered caches are now distributed **ASYNC** caches. The replicated caches are no longer even configured by default. See Configure the Cache Mode in the JBoss EAP *Configuration Guide* for information about how to add a replicated cache and make it the default.

This only affects you when you use the new JBoss EAP 7 default configuration. If you migrate the configuration from JBoss EAP 6, the configuration of the **infinispan** subsystem will be preserved.

### 4.7.2. Infinispan Cache Strategy Changes

The behavior of **ASYNC** cache strategy has changed in JBoss EAP 7.

In JBoss EAP 6, **ASYNC** cache reads were lock free. Although they would never block, the were prone to dirty reads of stale data, for example on failover. This is because it would allow subsequent requests for the same user to start before the previous request completed. This permissiveness is not acceptable when using distributed mode, since cluster topology changes can affect session affinity and easily result in stale data.

In JBoss EAP 7, **ASYNC** cache reads require locks. Since they now block new requests from the same user until the previous replication finishes, dirty reads are prevented.

### 4.7.3. Configuring Custom Stateful Session Bean Cache for Passivation

Be aware of the following restrictions when configuring a custom stateful session bean (SFSB) cache for passivation in JBoss EAP 7.1 and later.

- The **idle-timeout** attribute, which is configured in the **infinispan passivation-store** of the **ejb3** subsystem, is deprecated in JBoss EAP 7.1 and later. JBoss EAP 6.4 supported eager passivation, passivating according to the **idle-timeout** value. JBoss EAP 7.1 and later support lazy passivation, passivating when the **max-size** threshold is reached.

- In JBoss EAP 7.1 and later, the cluster name used by the EJB client is determined by the actual cluster name of the channel, as configured in the **jgroups** subsystem.

- JBoss EAP 7.1 and later still allow you to set the **max-size** attribute to control the passivation threshold.

- You should not configure eviction or expiration in your EJB cache configuration.
    - You should configure eviction by using the **max-size** attribute of the **passivation-store** in the **ejb3** subsystem.
    - You should configure expiration by using the **@StatefulTimeout** annotation in the SFSB Java source code or by specifying a **stateful-timeout** value in the **ejb-jar.xml** file.

### 4.7.4. Infinispan Cache Container Transport Changes

A change in behavior between JBoss EAP 7.0 and later versions requires that any updates to the cache container transport protocol to be done in batch mode or using a special header. This change in behavior also impacts any tools that are used to manage the JBoss EAP server.

The following is an example of the management CLI commands used to configure the cache container transport protocol in JBoss EAP 7.0.

```
/subsystem=infinispan/cache-container=my:add()
/subsystem=infinispan/cache-container=my/transport=jgroups:add()
/subsystem=infinispan/cache-container=my/invalidation-cache=mycache:add(mode=SYNC)
```

The following is an example of the management CLI commands needed to perform the same configuration in JBoss EAP 7.1. Note that the commands are executed in batch mode.

```
batch
/subsystem=infinispan/cache-container=my:add()
/subsystem=infinispan/cache-container=my/transport=jgroups:add()
/subsystem=infinispan/cache-container=my/invalidation-cache=mycache:add(mode=SYNC)
run-batch
```

If you prefer not to use batch mode, you can instead specify the operation header **allow-resource-service-restart=true** when defining the transport. Be aware that this restarts the service so that the operations can take effect, and some services might stop working until the service is restarted.

If you use scripts to update the cache container transport protocol, be sure to review them and add batch mode.

## 4.8. EJB SERVER CONFIGURATION CHANGES

There is no **migrate** operation for the **ejb3** subsystem, so if you use the management CLI **migrate** operations to upgrade your other existing JBoss EAP 6.4 configurations, be aware that the **ejb3** subsystem configuration is not migrated. Because the configuration of the **ejb3** subsystem is slightly different in JBoss EAP 7 than in JBoss EAP 6.4, you might see exceptions in the server log when you deploy your EJB applications.

IMPORTANT

If you use the JBoss Server Migration Tool to update your server configuration, the **ejb3** subsystem should be configured correctly and you should not see any issues when you deploy your EJB applications. For information about how to configure and run the tool, see Using the JBoss Server Migration Tool.

### DuplicateServiceException

The following **DuplicateServiceException** is caused by caching changes in JBoss EAP 7.

#### DuplicateServiceException in Server Log

```
ERROR [org.jboss.msc.service.fail] (MSC service thread 1-3) MSC000001: Failed to start service
jboss.deployment.unit."mdb-1.0-SNAPSHOT.jar".cache-dependencies-installer:
org.jboss.msc.service.StartException in service jboss.deployment.unit."mdb-1.0-
SNAPSHOT.jar".cache-dependencies-installer: Failed to start service
...
Caused by: org.jboss.msc.service.DuplicateServiceException: Service jboss.infinispan.ejb."mdb-1.0-
SNAPSHOT.jar".config is already registered
```

You must reconfigure the cache to resolve this error.

1. Follow the instructions to Start the Server and the Management CLI.

2. Issue the following commands to reconfigure caching in the **ejb3** subsystem.

```
/subsystem=ejb3/file-passivation-store=file:remove
/subsystem=ejb3/cluster-passivation-store=infinispan:remove
/subsystem=ejb3/passivation-store=infinispan:add(cache-container=ejb, max-size=10000)

/subsystem=ejb3/cache=passivating:remove
```

```
/subsystem=ejb3/cache=clustered:remove
/subsystem=ejb3/cache=distributable:add(passivation-store=infinispan, aliases=[passivating,
clustered])
```

## 4.9. MESSAGING SERVER CONFIGURATION CHANGES

In JBoss EAP 7, ActiveMQ Artemis replaces HornetQ as the JMS support provider. This section describes how to migrate the configuration and related messaging data.

### 4.9.1. Messaging Subsystem Server Configuration Changes

The **org.jboss.as.messaging** module extension, located in ***EAP_HOME*/modules/system/layers/base/**, has been replaced by the **org.wildfly.extension.messaging-activemq** extension module.

The **urn:jboss:domain:messaging:3.0** subsystem configuration namespace has been replaced by the **urn:jboss:domain:messaging-activemq:4.0** namespace.

#### Management Model

In most cases, an effort was made to keep the element and attribute names as similar as possible to those used in previous releases. The following table lists some of the changes.

**Table 4.3. Mapping Messaging Attributes**

| HornetQ Name | ActiveMQ Name |
|---|---|
| hornetq-server | server |
| hornetq-serverType | serverType |
| connectors | connector |
| discovery-group-name | discovery-group |

The management operations invoked on the new **messaging-activemq** subsystem have changed from **/subsystem=messaging/hornetq-server=** to **/subsystem=messaging-activemq/server=**.

You can migrate an existing JBoss EAP 6 **messaging** subsystem configuration to the **messaging-activemq** subsystem on a JBoss EAP 7 server by invoking its **migrate** operation.

```
/subsystem=messaging:migrate
```

Before you execute the **migrate** operation, you can invoke the **describe-migration** operation to review the list of management operations that will be performed to migrate from the existing JBoss EAP 6 **messaging** subsystem configuration to the **messaging-activemq** subsystem on the JBoss EAP 7 server.

```
/subsystem=messaging:describe-migration
```

The **migrate** and **describe-migration** operations also display a list of **migration-warnings** for resources or attributes that can not be migrated automatically.

#### Messaging Subsystem Migration and Forward Compatibility

The **describe-migration** and **migrate** operations for the **messaging** subsystem provide an additional configuration argument. If you want to configure messaging to allow legacy JBoss EAP 6 clients to connect to the JBoss EAP 7 server, you can add the boolean **add-legacy-entries** argument to the **describe-migration** or **migrate** operation as follows.

```
/subsystem=messaging:describe-migration(add-legacy-entries=true)
/subsystem=messaging:migrate(add-legacy-entries=true)
```

If the boolean argument **add-legacy-entries** is set to **true**, the **messaging-activemq** subsystem creates the **legacy-connection-factory** resource and adds **legacy-entries** to the **jms-queue** and **jms-topic** resources.

If the boolean argument **add-legacy-entries** is set to **false**, no legacy resources are created in the **messaging-activemq** subsystem and legacy JMS clients will not be able to communicate with the JBoss EAP 7 servers. This is the default value.

For more information about forward and backward compatibility see the Backward and Forward Compatibility in *Configuring Messaging* for JBoss EAP.

For more information about the management CLI **migrate** and **describe-migration** operations, see Management CLI Migration Operation.

#### Change in Behavior of forward-when-no-consumers Attribute

The behavior of the **forward-when-no-consumers** attribute has changed in JBoss EAP 7.

In JBoss EAP 6, when **forward-when-no-consumers** was set to **false** and there were no consumers in a cluster, messages were redistributed to all nodes in a cluster.

This behavior has changed in JBoss EAP 7. When **forward-when-no-consumers** is set to **false** and there are no consumers in a cluster, messages are not redistributed. Instead, they are kept on the original node to which they were sent.

#### Change in Default Cluster Load Balancing Policy

The default cluster load balancing policy has changed in JBoss EAP 7.

In JBoss EAP 6, the default cluster load balancing policy was similar to **STRICT**, which is like setting the legacy **forward-when-no-consumers** parameter to **true**. In JBoss EAP 7, the default is now **ON_DEMAND**, which is like setting the legacy **forward-when-no-consumers** parameter to **false**. For more information about these settings, see Cluster Connection Attributes in *Configuring Messaging* for JBoss EAP.

#### Messaging Subsystem XML Configuration

The XML configuration has changed significantly with the new **messaging-activemq** subsystem, and now provides an XML scheme more consistent with other JBoss EAP subsystems.

It is strongly advised that you do not attempt to modify the JBoss EAP **messaging** subsystem XML configuration to conform to the new **messaging-activemq** subsystem. Instead, invoke the legacy subsystem **migrate** operation. This operation will write the XML configuration of the new **messaging-activemq** subsystem as a part of its execution.

### 4.9.2. Migrate Messaging Data

You can use one of the following approaches to migrate messaging data from a previous release to the current release of JBoss EAP.

- For file-based messaging systems, you can migrate messaging data to JBoss EAP 7.2 from JBoss EAP 6.4 and previous JBoss EAP 7.x releases using the export and import method. With this method you export the messaging data from the previous release and import it using the management CLI **import-journal** operation. Be aware that you can use this approach for file-based messaging systems only.

- You can migrate messaging data from JBoss EAP 6.4 to JBoss EAP 7.2 by configuring a JMS bridge. You can use this approach for both file-based and JDBC messaging systems.

Due to the change from HornetQ to ActiveMQ Artemis as the JMS support provider, both the format and the location of the messaging data changed in JBoss EAP 7.0 and later. See Mapping Messaging Folder Names for details of the changes to the messaging data folder names and locations between the 6.4 and 7.x releases.

#### 4.9.2.1. Migrate Messaging Data Using Export and Import

Using this approach, you export the messaging data from a previous release to an XML file, and then import that file using the **import-journal** operation.

1. Export the messaging data to an XML file.
   - Export messaging data from JBoss EAP 6.4.
   - Export messaging data from JBoss EAP 7.x.
2. Import the XML formatted messaging data.

**IMPORTANT**

You cannot use the export and import method to move messaging data between systems that use a JDBC-based journal for storage.

**Export Messaging Data from JBoss EAP 6.4**

Due to the change from HornetQ to ActiveMQ Artemis as the JMS support provider, both the format and the location of the messaging data changed in JBoss EAP 7.0 and later.

To export messaging data from JBoss EAP 6.4, you must use the HornetQ **exporter** utility. The HornetQ **exporter** utility generates and exports the messaging data from JBoss EAP 6.4 to an XML file. This command requires that you specify the paths to the required HornetQ JARs that shipped with JBoss EAP 6.4, pass the paths to **messagingbindings**/, **messagingjournal**/, **messagingpaging**/, and **messaginglargemessages**/ folders from the previous release as arguments, and specify an output file in which to write the exported XML data.

The following is the syntax required by the HornetQ **exporter** utility.

```
$ java -jar -mp MODULE_PATH org.hornetq.exporter MESSAGING_BINDINGS_DIRECTORY
MESSAGING_JOURNAL_DIRECTORY MESSAGING_PAGING_DIRECTORY
MESSAGING_LARGE_MESSAGES_DIRECTORY > OUTPUT_DATA.xml
```

Create a custom module to ensure the correct versions of the HornetQ JARs, including any JARs installed with patches or upgrades, are loaded and made available to the **exporter** utility. Using your favorite editor, create a new **module.xml** file in the ***EAP6_HOME*/modules/org/hornetq/exporter/main/** directory and copy the following content:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<module xmlns="urn:jboss:module:1.1" name="org.hornetq.exporter">
   <main-class name="org.hornetq.jms.persistence.impl.journal.XmlDataExporter"/>
   <properties>
      <property name="jboss.api" value="deprecated"/>
   </properties>
   <dependencies>
      <module name="org.hornetq"/>
   </dependencies>
</module>
```

**NOTE**

The custom module is created in the **modules/** directory, not the **modules/system/layers/base/** directory.

Follow the steps below to export the data.

1. Stop the JBoss EAP 6.4 server.
2. Create the custom module as described above.
3. Run the following command to export the data.

   ```
   $ java -jar jboss-modules.jar -mp modules/ org.hornetq.exporter
   standalone/data/messagingbindings/ standalone/data/messagingjournal/
   standalone/data/messagingpaging standalone/data/messaginglargemessages/ >
   OUTPUT_DIRECTORY/OldMessagingData.xml
   ```

4. Make sure there are no errors or warning messages in the log at the completion of the command.
5. Use tooling available for your operating system to validate the XML in the generated output file.

**Export Messaging Data from JBoss EAP 7.x**
Follow these steps to export messaging data from JBoss EAP 7.x.

1. Open a terminal, navigate to the JBoss EAP 7.x install directory, and start the server in **admin-only** mode.

   ```
   $ EAP_HOME/bin/standalone.sh -c standalone-full.xml --start-mode=admin-only
   ```

2. Open a new terminal, navigate to the JBoss EAP 7.x install directory, and connect to the management CLI.

   ```
   $ EAP_HOME/bin/jboss-cli.sh --connect
   ```

3. Use the following management CLI command to export the messaging journal data.

   ```
   /subsystem=messaging-activemq/server=default:export-journal()
   ```

4. Make sure there are no errors or warning messages in the log at the completion of the command.
5. Use tooling available for your operating system to validate the XML in the generated output file.

**Import the XML Formatted Messaging Data**

You then import the XML file into JBoss EAP 7.0 or later by using the **import-journal** operation as follows.

> **IMPORTANT**
>
> If your target server has already performed some messaging tasks, be sure to back up your messaging folders before you begin the **import-journal** operation to prevent data loss in the event of an import failure. See Backing Up Messaging Folder Data for more information.

1. If you are migrating your JBoss EAP 6.4 server to JBoss EAP 7.2, make sure you have completed the migration of the server configuration before you begin by using the management CLI migrate operation or by running the JBoss Server Migration Tool. For information about how to configure and run the tool, see Using the JBoss Server Migration Tool.

2. Start the JBoss EAP 7.x server in normal mode with *no* JMS clients connected.

   > **IMPORTANT**
   >
   > It is important that you start the server with no JMS clients connected. This is because the **import-journal** operation behaves like a JMS producer. Messages are immediately available when the operation is in progress. If this operation fails in the middle of the import and JMS clients are connected, there is no way to recover because JMS clients might have already consumed some of the messages.

3. Open a new terminal, navigate to the JBoss EAP 7.x install directory, and connect to the management CLI.

   ```
   $ EAP_HOME/bin/jboss-cli.sh --connect
   ```

4. Use the following management CLI command to import the messaging data.

   ```
   /subsystem=messaging-activemq/server=default:import-
   journal(file=OUTPUT_DIRECTORY/OldMessagingData.xml)
   ```

   > **IMPORTANT**
   >
   > Do *not* run this command more than one time as doing so will result in duplicate messages!

WARNING

If you are using JBoss EAP 7.0, you must apply Red Hat JBoss Enterprise Application Platform 7.0 Update 05 or a newer cumulative patch to your JBoss EAP installation in order to avoid a known issue when reading large messages. For more information, see JBEAP-4407 - Consumer crashes with IndexOutOfBoundsException when reading large messages from imported journal.

This issue does not affect JBoss EAP 7.1 and later.

**Recovering from an Import Messaging Data Failure**
If the **import-journal** operation fails, you can attempt to recover by using the following steps.

1. Shut down the JBoss EAP 7.x server.
2. Delete all of the messaging journal folders. See Backing Up Messaging Folder Data for the management CLI commands to determine the correct directory location for the messaging journal folders.
3. If you backed up the target server messaging data prior to the import, copy the messaging folders from the backup location to the messaging journal directory determined in the prior step.
4. Repeat the steps to import the XML formatted messaging data.

### 4.9.2.2. Migrate Messaging Data Using a JMS Bridge

Using this approach, you configure and deploy a JMS bridge to the JBoss EAP 7.x server. The JMS bridge moves messages from the JBoss EAP 6.4 HornetQ queue to the JBoss EAP 7.x ActiveMQ Artemis queue.

A JMS bridge consumes messages from a source JMS queue or topic and sends them to a target JMS queue or topic, which is typically on a different server. It can be used to bridge messages between any JMS servers, as long as they are JMS 1.1 compliant. The source and destination JMS resources are looked up using JNDI and the client classes for the JNDI lookup must be bundled in a module. The module name is then declared in the JMS bridge configuration.

This section describes how to configure the servers and deploy a JMS bridge to move the messaging data from JBoss EAP 6.4 to JBoss EAP 7.x.

1. Configure the source JBoss EAP 6.4 server.
2. Configure the target JBoss EAP 7.x server.
3. Migrate the messaging data.

**Configure the Source JBoss EAP 6.4 Server**

1. Stop the JBoss EAP 6.4 server.
2. Back up the HornetQ journal and configuration files.

- By default, the HornetQ journal is located in the ***EAP6_HOME*/standalone/data/** directory.
- See Mapping Messaging Folder Names for default messaging folder locations for each release.

3. Make sure that the **InQueue** JMS queue containing the JMS messages is defined on the JBoss EAP 6.4 server.

4. Make sure that **messaging** subsystem configuration contains an entry for the **RemoteConnectionFactory** similar to the following.

```
<connection-factory name="RemoteConnectionFactory">
    <entries>
        <entry name="java:jboss/exported/jms/RemoteConnectionFactory"/>
    </entries>
    ...
</connection-factory>
```

If it does not contain the entry, create one using the following management CLI command:

```
/subsystem=messaging/hornetq-server=default/connection-
factory=RemoteConnectionFactory:add(factory-type=XA_GENERIC, connector=[netty],
entries=[java:jboss/exported/jms/RemoteConnectionFactory],ha=true,block-on-
acknowledge=true,retry-interval=1000,retry-interval-multiplier=1.0,reconnect-attempts=-1)
```

### Configure the Target JBoss EAP 7.x Server

1. The JMS bridge configuration needs the **org.hornetq** module to connect to the HornetQ server in the previous release. This module and its direct dependencies are not present in JBoss EAP 7.x, so you must copy the following modules from the previous release.

    - Copy the **org.hornetq** module into the JBoss EAP 7.x ***EAP_HOME*/modules/org/** directory.
        - If you did not apply patches to this module, copy this folder from the JBoss EAP 6.4 server: ***EAP6_HOME*/modules/system/layers/base/org/hornetq/**
        - If you did apply patches to this module, copy this folder from the JBoss EAP 6.4 server: ***EAP6_HOME*/modules/system/layers/base/.overlays/layer-base-jboss-eap-6.4.x.CP/org/hornetq/**
    - Remove the **<resource-root>** for the HornetQ **lib** path from the JBoss EAP 7.x ***EAP_HOME*/modules/org/hornetq/main/module.xml** file.
        - If you did not apply patches to the JBoss EAP 6.4 **org.hornetq** module, remove the following line from the file:

        ```
        <resource-root path="lib"/>
        ```

        - If you did apply patches to the JBoss EAP 6.4 **org.hornetq** module, remove the following lines from the file:

        ```
        <resource-root path="lib"/>
        <resource-root path="../../../../../org/hornetq/main/lib"/>
        ```

> **WARNING**
>
> Failure to remove the HornetQ **lib** path **resource-root** will cause the bridge to fail with the following error in the log file.
>
> ```
> 2016-07-15 09:32:25,660 ERROR
> [org.jboss.as.controller.management-operation] (management-
> handler-thread - 2) WFLYCTL0013: Operation ("add") failed -
> address: ([
>     ("subsystem" => "messaging-activemq"),
>     ("jms-bridge" => "myBridge")
> ]) - failure description: "WFLYMSGAMQ0086: Unable to load
> module org.hornetq"
> ```

- Copy the **org.jboss.netty** module into the JBoss EAP 7.x ***EAP_HOME*/modules/org/jboss/** directory.
  - If you did not apply patches to this module, copy this folder from the JBoss EAP 6.4 server: ***EAP6_HOME*/modules/system/layers/base/org/jboss/netty/**
  - If you did apply patches to this module, copy this folder from the JBoss EAP 6.4 server: ***EAP6_HOME*/modules/system/layers/base/.overlays/layer-base-jboss-eap-6.4.x.CP/org/jboss/netty**

2. Create the JMS queue to contain the messages received from JBoss EAP 6.4 server. The following is an example of a management CLI command that creates the **MigratedMessagesQueue** JMS queue to receive the message.

```
jms-queue add --queue-address=MigratedMessagesQueue --entries=
[jms/queue/MigratedMessagesQueue
java:jboss/exported/jms/queue/MigratedMessagesQueue]
```

This creates the following **jms-queue** configuration for the default server in the **messaging-activemq** subsystem of the JBoss EAP 7.x server.

```
<jms-queue name="MigratedMessagesQueue" entries="jms/queue/MigratedMessagesQueue
java:jboss/exported/jms/queue/MigratedMessagesQueue"/>
```

3. Make sure that **messaging-activemq** subsystem **default** server contains a configuration for the **InVmConnectionFactory connection-factory** similar to the following:

```
<connection-factory name="InVmConnectionFactory" factory-type="XA_GENERIC"
entries="java:/ConnectionFactory" connectors="in-vm"/>
```

If it does not contain the entry, create one using the following management CLI command:

```
/subsystem=messaging-activemq/server=default/connection-
factory=InVmConnectionFactory:add(factory-type=XA_GENERIC, connectors=[in-vm],
entries=[java:/ConnectionFactory])
```

4. Create and deploy a JMS bridge that reads messages from the **InQueue** JMS queue configured on the JBoss EAP 6.4 server and transfers them to the **MigratedMessagesQueue** configured on the JBoss EAP 7.x server.

```
/subsystem=messaging-activemq/jms-bridge=myBridge:add(add-messageID-in-
header=true,max-batch-time=100,max-batch-size=10,max-retries=-1,failure-retry-
interval=1000,quality-of-service=AT_MOST_ONCE,module=org.hornetq,source-
destination=jms/queue/InQueue,source-connection-
factory=jms/RemoteConnectionFactory,source-context=
[("java.naming.factory.initial"=>"org.wildfly.naming.client.WildFlyInitialContextFactory"),
("java.naming.provider.url"=>"remote://127.0.0.1:4447")],target-
destination=jms/queue/MigratedMessagesQueue,target-connection-
factory=java:/ConnectionFactory)
```

This creates the following **jms-bridge** configuration in the **messaging-activemq** subsystem of the JBoss EAP 7.x server.

```
<jms-bridge name="myBridge" add-messageID-in-header="true" max-batch-time="100" max-
batch-size="10" max-retries="-1" failure-retry-interval="1000" quality-of-
service="AT_MOST_ONCE" module="org.hornetq">
    <source destination="jms/queue/InQueue" connection-
factory="jms/RemoteConnectionFactory">
        <source-context>
            <property name="java.naming.factory.initial"
value="org.wildfly.naming.client.WildFlyInitialContextFactory"/>
            <property name="java.naming.provider.url" value="remote://127.0.0.1:4447"/>
        </source-context>
    </source>
    <target destination="jms/queue/MigratedMessagesQueue" connection-
factory="java:/ConnectionFactory"/>
</jms-bridge>
```

5. If security is configured for JBoss EAP 6.4, you must also configure the JMS bridge configuration **<source>** element to include a **source-context** that specifies the correct user name and password to use for the JNDI lookup when creating the connection.

**Migrate the Messaging Data**

1. Verify that the information you provided for the following configurations is correct.
    - Any queue and topic names.
    - The **java.naming.provider.url** for JNDI lookup.
2. Make sure that you have deployed the target JMS destination to the JBoss EAP 7.x server.
3. Start both the JBoss EAP 6.4 and JBoss EAP 7.x servers.

#### 4.9.2.3. Mapping Messaging Folder Names

The following table shows the messaging directory names used in the previous release and the corresponding names used in the current release of JBoss EAP. The directories are relative to the **jboss.server.data.dir** directory, which defaults to ***EAP_HOME*/standalone/data/** if it is not specified.

| JBoss EAP 6.4 Directory Name | JBoss EAP 7.x Directory Name |
|---|---|
| **messagingbindings/** | **activemq/bindings/** |
| **messagingjournal/** | **activemq/journal/** |
| **messaginglargemessages/** | **activemq/largemessages/** |
| **messagingpaging/** | **activemq/paging/** |

NOTE

The **messaginglargemessages/** and **messagingpaging/** directories might not be present if there are no large messages or if paging is disabled.

#### 4.9.2.4. Backing Up Messaging Folder Data

If your target server has already processed messages, it is a good idea to back up the target message folders to a backup location before you begin. The default location of the messaging folders is ***EAP_HOME*/standalone/data/activemq/**; however it is configurable. If you are not sure of the location of your messaging data, you can use the following management CLI commands to find the location of the messaging folders.

```
/subsystem=messaging-activemq/server=default/path=journal-directory:resolve-path
/subsystem=messaging-activemq/server=default/path=paging-directory:resolve-path
/subsystem=messaging-activemq/server=default/path=bindings-directory:resolve-path
/subsystem=messaging-activemq/server=default/path=large-messages-directory:resolve-path
```

Once you know the location of the folders, copy each folder to a safe backup location.

### 4.9.3. Migrate JMS Destinations

In JBoss EAP 6, JMS destination queues were configured in the **<jms-destinations>** element under the **<hornetq-server>** element in the **messaging** subsystem.

```
<hornetq-server>
  ...
  <jms-destinations>
     <jms-queue name="testQueue">
        <entry name="queue/test"/>
         <entry name="java:jboss/exported/jms/queue/test"/>
      </jms-queue>
  </jms-destinations>
  ...
</hornetq-server>
```

In JBoss EAP 7, the JMS destination queue is configured in the default **<server>** element of the **messaging-activemq** subsystem.

```
<server name="default">
  ...
  <jms-queue name="testQueue" entries="queue/test java:jboss/exported/jms/queue/test"/>
  ...
</server>
```

### 4.9.4. Migrate Messaging Interceptors

Messaging interceptors have changed significantly in JBoss EAP 7 with the replacement of HornetQ with ActiveMQ Artemis as the JMS messaging provider.

The HornetQ **messaging** subsystem included in the previous release of JBoss EAP required that you install the HornetQ interceptors by adding them to a JAR and then modifying the HornetQ **module.xml** file.

The **messaging-activemq** subsystem included in JBoss EAP 7 does not require modification of a **module.xml** file. User interceptor classes, which now implement the Apache ActiveMQ Artemis Interceptor interface, can now be loaded from any server module. You specify the module from which the interceptor should be loaded in the **messaging-activemq** subsystem of the server configuration file.

#### Example: Interceptor Configuration

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
  <server name="default">
    ...
    <incoming-interceptors>
      <class name="com.mycompany.incoming.myInterceptor" module="com.mycompany" />
      <class name="com.othercompany.incoming.myOtherInterceptor" module="com.othercompany" />
    </incoming-interceptors>
    <outgoing-interceptors>
      <class name="com.mycompany.outgoing.myInterceptor" module="com.mycompany" />
      <class name="com.othercompany.outgoing.myOtherInterceptor" module="com.othercompany" />
    </outgoing-interceptors>
  </server>
</subsystem>
```

### 4.9.5. Replace Netty Servlet Configuration

In JBoss EAP 6, you could configure a servlet engine to work with the Netty Servlet transport. Because ActiveMQ Artemis replaces HornetQ as the built-in messaging provider in JBoss EAP 7, this configuration is no longer available. You must replace the servlet configuration to use the new built-in messaging HTTP connectors and HTTP acceptors instead.

### 4.9.6. Configuring a Generic JMS Resource Adapter

The way you configure a generic JMS resource adapter for use with a third-party JMS provider has changed in JBoss EAP 7. For more information, see Deploying a Generic JMS Resource Adapter in *Configuring Messaging* for JBoss EAP.

### 4.9.7. Messaging Configuration Changes

In JBoss EAP 7.0, if you configured the **replication-master** policy without specifying the **check-for-live-server** attribute, its default value was **false**. This has changed in JBoss EAP 7.1 and later. The default value for the **check-for-live-server** attribute is now **true**.

The following is an example of a management CLI command that configures the **replication-master** policy without specifying the **check-for-live-server** attribute.

```
/subsystem=messaging-activemq/server=default/ha-policy=replication-master:add(cluster-name=my-cluster,group-name=group1)
```

When you read the resource using the management CLI, note that the **check-for-live-server** attribute value is set to **true**.

```
/subsystem=messaging-activemq/server=default/ha-policy=replication-master:read-resource(recursive=true)
{
    "outcome" => "success",
    "result" => {
        "check-for-live-server" => true,
        "cluster-name" => "my-cluster",
        "group-name" => "group1",
        "initial-replication-sync-timeout" => 30000L
    },
    "response-headers" => {"process-state" => "reload-required"}
}
```

### 4.9.8. Changes in JMS Serialization Behavior Between Releases

The **serialVersionUID** of **javax.jms.JMSException** changed between JMS 1.1 and JMS 2.0.0. This means that if an instance of a **JMSException**, or any of its subclasses, is serialized using JMS 1.1, it cannot be deserialized using JMS 2.0.0. The reverse is also true. If an instance of **JMSException** is serialized using JMS 2.0.0, it cannot be deserialized using JMS 1.1. In both of these cases, it throws an exception similar to the following:

```
javax.jms.JMSException: javax.jms.JMSException; local class incompatible: stream classdesc serialVersionUID = 8951994251593378324, local class serialVersionUID = 2368476267211489441
```

This issue is fixed in the JMS 2.0.1 maintenance release.

The following table details the JMS implementation for each JBoss EAP release.

**Table 4.4. JMS Implementation for Each JBoss EAP Release**

| JBoss EAP Version | JMS Implementation | JMS version |
|---|---|---|
| 6.4 | HornetQ | JMS 1.1 |
| 7.0 | Apache ActiveMQ Artemis | JMS 2.0.0 |
| 7.1 and later | Apache ActiveMQ Artemis | JMS 2.0.1 or later |

Be aware that the **serialVersionUID** incompatibility can result in a migration issue in the following situations:

- If you send a message that contains a **JMSException** using a JBoss EAP 6.4 client, migrate your messaging data to JBoss EAP 7.0, and then attempt to deserialize that message using a JBoss

EAP 7.0 client, the deserialization will fail and it will throw an exception. This is because the **serialVersionUID** in JMS 1.1 is *not* compatible with the one in JMS 2.0.0.

- If you send a message that contains a **JMSException** using a JBoss EAP 7.0 client, migrate your messaging data to JBoss EAP 7.1 or later, and then attempt to deserialize that message using a JBoss EAP 7.1 or later client, the deserialization will fail and it will throw an exception. This is because the **serialVersionUID** in JMS 2.0.0 is *not* compatible with the one in JMS 2.0.1 or later.

Note that if you send a message that contains a **JMSException** using a JBoss EAP 6.4 client, migrate your messaging data to JBoss EAP 7.1 or later, and then attempt to deserialize that message using a JBoss EAP 7.1 or later client, the deserialization will succeed because the **serialVersionUID** in JMS 1.1 is compatible with the one in JMS 2.0.1 or later.

IMPORTANT

Red Hat recommends that you do the following before you migrate your messaging data:

- Be sure to consume all JMS 1.1 messages that contain JMSExceptions before migrating messaging data from JBoss EAP 6.4 to JBoss EAP 7.0.
- Be sure to consume all JMS 2.0.0 messages that contain JMSExceptions before migrating messaging data from JBoss EAP 7.0 to JBoss EAP 7.1 or later.

## 4.10. JMX MANAGEMENT CHANGES

The HornetQ component in JBoss EAP 6 provided its own JMX management; however, it was not recommended and is now deprecated and no longer supported. If you relied on this feature in JBoss EAP 6, you must migrate your management tooling to use either the JBoss EAP management CLI or the JMX management provided with JBoss EAP 7.

You must also upgrade your client libraries to use the **jboss-client.jar** that ships with JBoss EAP 7.

The following is an example of HornetQ JMX management code that was used in JBoss EAP 6.

```
JMXConnector connector = null;
try {
   HashMap environment = new HashMap();
   String[] credentials = new String[]{"admin", "Password123!"};
   environment.put(JMXConnector.CREDENTIALS, credentials);

   // HornetQ used the protocol "remoting-jmx" and port "9999"
   JMXServiceURL beanServerUrl = new JMXServiceURL("service:jmx:remoting-
jmx://127.0.0.1:9990");

   connector = JMXConnectorFactory.connect(beanServerUrl, environment);
   MBeanServerConnection mbeanServer = connector.getMBeanServerConnection();

   // The JMX object name pointed to the HornetQ JMX management
   ObjectName objectName = new ObjectName("org.hornetq:type=Server,module=JMS");

   // The invoked method name was "listConnectionIDs"
   String[] connections = (String[]) mbeanServer.invoke(objectName, "listConnectionIDs", new
Object[]{}, new String[]{});
   for (String connection : connections) {
      System.out.println(connection);
```

```
    }
} finally {
    if (connector != null) {
      connector.close();
    }
}
```

The following is an example of the equivalent code needed for ActiveMQ Artemis in JBoss EAP 7.

```
JMXConnector connector = null;
try {
    HashMap environment = new HashMap();
    String[] credentials = new String[]{"admin", "Password123!"};
    environment.put(JMXConnector.CREDENTIALS, credentials);

    // ActiveMQ Artemis uses the protocol "remote+http" and port "9990"
    JMXServiceURL beanServerUrl = new
JMXServiceURL("service:jmx:remote+http://127.0.0.1:9990");

    connector = JMXConnectorFactory.connect(beanServerUrl, environment);
    MBeanServerConnection mbeanServer = connector.getMBeanServerConnection();

    // The JMX object name points to the new JMX management in the `messaging-activemq`
subsystem
    ObjectName objectName = new ObjectName("jboss.as:subsystem=messaging-
activemq,server=default");

    // The invoked method name is now "listConnectionIds"
    String[] connections = (String[]) mbeanServer.invoke(objectName, "listConnectionIds", new Object[]
{}, new String[]{});
    for (String connection : connections) {
        System.out.println(connection);
    }
} finally {
    if (connector != null) {
      connector.close();
    }
}
```

Notice that the method names and parameters have changed in the new implementation. You can find the new method names in the JConsole by following these steps.

1. Connect to the JConsole using the following command.

   ```
   $ EAP_HOME/bin/jconsole.sh
   ```

2. Connect to JBoss EAP local process. Note that it should start with "jboss-modules.jar".

3. In the **MBeans** tab, choose **jboss.as → messaging-activemq → default → Operations** to display the list of method names and attributes.

## 4.11. ORB SERVER CONFIGURATION CHANGES

The JacORB implementation has been replaced with a downstream branch of the OpenJDK ORB in JBoss EAP 7.

The **org.jboss.as.jacorb** extension module, located in ***EAP_HOME*/modules/system/layers/base/**, has been replaced by the **org.wildfly.iiop-openjdk** extension module.

The **urn:jboss:domain:jacorb:1.4** subsystem configuration namespace in the server configuration file has been replaced by the **urn:jboss:domain:iiop-openjdk:2.1** namespace.

The following is an example of the default **jacorb** system configuration in JBoss EAP 6.

```
<subsystem xmlns="urn:jboss:domain:jacorb:1.4">
    <orb socket-binding="jacorb" ssl-socket-binding="jacorb-ssl">
        <initializers security="identity" transactions="spec"/>
    </orb>
</subsystem>
```

The following is an example of the default **iiop-openjdk** subsystem configuration in JBoss EAP 7.

```
<subsystem xmlns="urn:jboss:domain:iiop-openjdk:2.1">
    <orb socket-binding="jacorb" ssl-socket-binding="jacorb-ssl" />
    <initializers security="identity" transactions="spec" />
</subsystem>
```

The new **iiop-openjdk** subsystem configuration accepts only a subset of the legacy elements and attributes. The following is an example of a **jacorb** subsystem configuration in the previous release of JBoss EAP that contains all valid elements and attributes:

```
<subsystem xmlns="urn:jboss:domain:jacorb:1.4">
   <orb name="JBoss" print-version="off" use-imr="off" use-bom="off"  cache-typecodes="off"
       cache-poa-names="off" giop-minor-version="2" socket-binding="jacorb" ssl-socket-
binding="jacorb-ssl">
       <connection retries="5" retry-interval="500" client-timeout="0" server-timeout="0"
           max-server-connections="500" max-managed-buf-size="24" outbuf-size="2048"
           outbuf-cache-timeout="-1"/>
       <initializers security="off" transactions="spec"/>
   </orb>
   <poa monitoring="off" queue-wait="on" queue-min="10" queue-max="100">
       <request-processors pool-size="10" max-threads="32"/>
   </poa>
   <naming root-context="JBoss/Naming/root" export-corbaloc="on"/>
   <interop sun="on" comet="off" iona="off" chunk-custom-rmi-valuetypes="on"
       lax-boolean-encoding="off" indirection-encoding-disable="off" strict-check-on-tc-creation="off"/>
   <security support-ssl="off" add-component-via-interceptor="on" client-supports="MutualAuth"
       client-requires="None" server-supports="MutualAuth" server-requires="None"/>
   <properties>
       <property name="some_property" value="some_value"/>
   </properties>
</subsystem>
```

The following element attributes are no longer supported and must be removed.

**Table 4.5. Attributes to Remove**

| Element | Unsupported Attributes |
|---|---|

| Element | Unsupported Attributes |
|---|---|
| <orb> | • client-timeout<br>• max-managed-buf-size<br>• max-server-connections<br>• outbuf-cache-timeout<br>• outbuf-size<br>• connection retries<br>• retry-interval<br>• name<br>• server-timeout |
| <poa> | • queue-min<br>• queue-max<br>• pool-size<br>• max-threads |

The following **on/off** attributes are no longer supported and will not be migrated when you run the management CLI **migrate** operation. If they are set to **on**, you will get a migration warning. Other **on/off** attributes that are not mentioned in this table, for example **<security support-ssl="on|off">**, are still supported and will be migrated successfully. The only difference is that their values will be changed from **on/off** to **true/false**.

Table 4.6. Attributes to Turn Off or Remove

| Element | Attributes to Set to Off |
|---|---|
| <orb> | • cache-poa-names<br>• cache-typecodes<br>• print-version<br>• use-bom<br>• use-imr |

| Element | Attributes to Set to Off |
|---|---|
| <interop> | (all except **sun**)<br>● comet<br>● iona<br>● chunk-custom-rmi-valuetypes<br>● indirection-encoding-disable<br>● lax-boolean-encoding<br>● strict-check-on-tc-creation |
| <poa> | ● monitoring<br>● queue-wait |

## 4.12. MIGRATE THE THREADS SUBSYSTEM CONFIGURATION

The JBoss EAP 6 server configuration included a **threads** subsystem that was used to manage thread pools across the various subsystems in the server.

The **threads** subsystem is no longer available in JBoss EAP 7. Instead, each subsystem is responsible for managing its own thread pools.

For information about how to configure thread pools for the **infinispan** subsystem, see Configure Infinispan Thread Pools in the JBoss EAP *Configuration Guide*.

For information about how to configure thread pools for the **jgroups** subsystem, see Configure JGroups Thread Pools in the JBoss EAP *Configuration Guide*.

In JBoss EAP 6, you configured thread pools for connectors and listeners for the **web** subsystem by referencing an **executor** that was defined in the **threads** subsystem. In JBoss EAP 7, you now configure thread pools for the **undertow** subsystem by referencing a **worker** that is defined in the **io** subsystem. For more information, see Configuring the IO Subsystem in the JBoss EAP *Configuration Guide*.

For information about about changes to thread pool configuration in the **remoting** subsystem, see Migrate the Remoting Subsystem Configuration in this guide, and Configuring the Endpoint in the JBoss EAP *Configuration Guide*.

## 4.13. MIGRATE THE REMOTING SUBSYSTEM CONFIGURATION

In JBoss EAP 6, you configured the thread pool for the **remoting** subsystem by setting various **worker-*** attributes. The worker thread pool is no longer configured in the **remoting** subsystem in JBoss EAP 7 and if you attempt to modify the existing configuration, you will see the following message.

```
WFLYRMT0022: Worker configuration is no longer used, please use endpoint worker configuration
```

In JBoss EAP 7, the worker thread pool is replaced by an endpoint configuration that references a **worker** defined in the **io** subsystem.

For information about how to configure the endpoint, see Configuring the Endpoint in the JBoss EAP *Configuration Guide*.

## 4.14. WEBSOCKET SERVER CONFIGURATION CHANGES

To use WebSockets in JBoss EAP 6, you had to enable the non blocking Java NIO2 connector protocol for the **http** connector in the **web** subsystem of the JBoss EAP server configuration file using a command similar to the following.

```
/subsystem=web/connector=http/:write-
attribute(name=protocol,value=org.apache.coyote.http11.Http11NioProtocol)
```

To use WebSockets in an application, you also had to create a **<enable-websockets>** element in the application **WEB-INF/jboss-web.xml** file and set it to **true**.

In JBoss EAP 7, you no longer need to configure the server for default WebSocket support or configure the application to use it. WebSockets are a requirement of the Java EE 7 specification and the required protocols are configured by default. More complex WebSocket configuration is done in the **servlet-container** of the **undertow** subsystem of the JBoss EAP server configuration file. You can view the available settings using the following command.

```
/subsystem=undertow/servlet-container=default/setting=websockets:read-resource(recursive=true)
{
  "outcome" => "success",
  "result" => {
     "buffer-pool" => "default",
     "dispatch-to-worker" => true,
     "worker" => "default"
  }
}
```

For more information about WebSocket development, see Creating WebSocket Applications in the JBoss EAP *Development Guide*.

WebSocket code examples can also be found in the quickstarts that ship with JBoss EAP.

## 4.15. SINGLE SIGN-ON SERVER CHANGES

The **infinispan** subsystem still provides distributed caching support for HA services in the form of Infinispan caches in JBoss EAP 7; however the caching and distribution of authentication information is handled differently than in previous releases.

In JBoss EAP 6, if single sign-on (SSO) was not provided an Infinispan cache, the cache was not distributed.

In JBoss EAP 7, SSO is distributed automatically when you select the HA profile. When running the HA profile, each host has its own Infinispan cache, which is based on the default cache of the web cache container. This cache stores the relevant session and SSO cookie information for the host. JBoss EAP handles propagation of individual cache information to all hosts. There is no way to specifically assign an Infinispan cache to SSO in JBoss EAP 7.

In JBoss EAP 7, SSO is configured in the **undertow** subsystem of the server configuration file.

There are no application code changes required for SSO when migrating to JBoss EAP 7.

## 4.16. DATASOURCE CONFIGURATION CHANGES

### 4.16.1. JDBC Datasource Driver Name

When you configured a datasource in the previous release of JBoss EAP, the value specified for the driver name depended on the number of classes listed in the **META-INF/services/java.sql.Driver** file contained in the JDBC driver JAR.

**Driver Containing a Single Class**
If the **META-INF/services/java.sql.Driver** file specified only one class, the driver name value was simply the name of the JDBC driver JAR. This has not changed in JBoss EAP 7.

**Driver Containing Multiple Classes**
IIn JBoss EAP 6, if there was more than one class listed in **META-INF/services/java.sql.Driver** file, you specified which class was the driver class by appending its name to the JAR name, along with the major and minor version, in the following format.

```
JAR_NAME + DRIVER_CLASS_NAME + "_" + MAJOR_VERSION + "_" + MINOR_VERSION
```

In JBoss EAP 7, this has changed. You now specify the driver name using the following format.

```
JAR_NAME + "_" + DRIVER_CLASS_NAME + "_" + MAJOR_VERSION + "_" + MINOR_VERSION
```

NOTE

An underscore has been added between the *JAR_NAME* and the *DRIVER_CLASS_NAME*.

The MySQL 5.1.31 JDBC driver is an example of a driver that contains two classes. The driver class name is **com.mysql.jdbc.Driver**. The following examples demonstrate the differences between how you specify the driver name in the previous and current release of JBoss EAP.

**Example: JBoss EAP 6 Driver Name**

```
mysql-connector-java-5.1.31-bin.jarcom.mysql.jdbc.Driver_5_1
```

**Example: JBoss EAP 7 Driver Name**

```
mysql-connector-java-5.1.31-bin.jar_com.mysql.jdbc.Driver_5_1
```

## 4.17. SECURITY SERVER CONFIGURATION CHANGES

If you migrate to JBoss EAP 7 and plan to run with the Java Security Manager enabled, you should be aware that changes were made in the way policies are defined and that additional configuration changes might be needed. Also be aware that custom security managers are not supported in JBoss EAP 7.

For information about Java Security Manager server configuration changes, see Considerations Moving from Previous Versions in *How to Configure Server Security* for JBoss EAP.

### 4.17.1. Changes in Legacy Security Behavior between JBoss EAP 7.0 and JBoss EAP 7.1

#### 4.17.1.1. HTTP Status Change for Unreachable LDAP Realms

If no LDAP realm was reachable by the server in JBoss EAP 7.0, the **security** subsystem returned an HTTP status code of "401 Unauthorized".

The legacy **security** subsystem in JBoss EAP 7.1 and later instead return an HTTP status code of "500 Internal Error" to more accurately describe that an unexpected condition occurred that prevented the server from successfully processing the request.

#### 4.17.1.2. Enabling the LDAP Security Realm to Parse Roles from a DN

In JBoss EAP 7.0, the **org.jboss.as.domain.management.security.parseGroupNameFromLdapDN** system property was used to enable the LDAP security realm to parse for roles from a DN. When this property was set to **true**, roles were parsed from a DN. Otherwise, a normal LDAP search was used to search for roles.

In JBoss EAP 7.1 and later, this system property is deprecated. Instead, you configure this option by setting the newly introduced **parse-group-name-from-dn** attribute to **true** in the core service path using the following management CLI command:

```
/core-service=management/security-realm=REALM_NAME/authorization=ldap/group-
search=principal-to-group:add(parse-group-name-from-dn=true)
```

#### 4.17.1.3. Changes in Sending the JBoss EAP SSL Certificate to an LDAP Server

In JBoss EAP 7.0, when the management interface is configured to use the **ldapSSL** security realm, mutual authentication between the server and LDAP can fail, resulting in an authentication failure in the management interface. This is because two different LDAP connections are made, each by a different thread, and they do not share the SSL sessions.

JBoss EAP 7.1 introduced a new boolean **always-send-client-cert** management attribute on the LDAP **outbound-connection**. This option allows configuration of outbound LDAP connections to support LDAP servers that are configured to always require a client certificate.

LDAP authentication happens in two steps:

1. It searches for the account.
2. It verifies the credentials.

By default, the **always-send-client-cert** attribute is set to **false**, meaning the client SSL certificate is sent only with the first search account request. When this attribute is set to **true**, the JBoss EAP LDAP client sends the client certificate to the LDAP server with both the search and verification requests.

You can set this attribute to **true** using the following management CLI command.

```
/core-service=management/ldap-connection=my-ldap-connection:write-attribute(name=always-send-
client-cert,value=true)
```

This results in the following LDAP outbound connection in the server configuration file.

```
<management>
  ....
  <outbound-connections>
    <ldap name="my-ldap-connection" url="ldap://127.0.0.1:389" search-
dn="cn=search,dc=myCompany,dc=com" search-credential="myPass" always-send-client-
cert="true"/>
  </outbound-connections>
  ....
</management>
```

### 4.17.2. FIPS Mode Changes

If you are running in FIPS mode, be aware that the default behavior has changed between JBoss EAP 7.0 and JBoss EAP 7.1.

When using legacy security realms, JBoss EAP 7.1 and later provide the automatic generation of a self-signed certificate for development purposes. This feature, which was not available in JBoss EAP 7.0, is enabled by default. This means that if you are running in FIPS mode, you must configure the server to disable automatic self-signed certificate creation. Otherwise, you might see the following error when you start the server.

```
ERROR [org.xnio.listener] (default I/O-6) XNIO001007: A channel event listener threw an exception:
java.lang.RuntimeException: WFLYDM0114: Failed to lazily initialize SSL context
...
Caused by: java.lang.RuntimeException: WFLYDM0112: Failed to generate self signed certificate
...
Caused by: java.security.KeyStoreException: Cannot get key bytes, not PKCS#8 encoded
```

For information about automatic self-signed certificate creation, see Automatic Self-signed Certificate Creation for Applications in *How to Configure Server Security* for JBoss EAP.

## 4.18. TRANSACTIONS SUBSYSTEM CHANGES

Some Transaction Manager configuration attributes that were available in the **transactions** subsystem in JBoss EAP 6 have changed in JBoss EAP 7.

### Removed Transactions Subsystem Attributes

The following table lists the JBoss EAP 6 attributes that were removed from the **transactions** subsystem in JBoss EAP 7 and the equivalent replacement attributes.

| Attribute in JBoss EAP 6 | Replacement in JBoss EAP 7 |
|---|---|
| path | object-store-path |
| relative-to | object-store-relative-to |

### Deprecated Transactions Subsystem Attributes

The following attributes that were available in the **transactions** subsystem in JBoss EAP 6 are deprecated in JBoss EAP 7. The deprecated attributes might be removed in a future release of the product. The following table lists the equivalent replacement attributes.

| Attribute in JBoss EAP 6 | Replacement in JBoss EAP 7 |
|---|---|
| use-hornetq-store | use-journal-store |
| hornetq-store-enable-async-io | journal-store-enable-async-io |
| enable-statistics | statistics-enabled |

## 4.19. CHANGES TO MOD_CLUSTER CONFIGURATION

The configuration for static proxy lists in mod_cluster has changed in JBoss EAP 7.

In JBoss EAP 6, you configured the **proxy-list** attribute, which was a comma-separated list of httpd proxy addresses specified in the format of **hostname:port**.

The **proxy-list** attribute is deprecated in JBoss EAP 7. It has been replaced by the **proxies** attribute, which is a list of outbound socket binding names.

This change impacts how you define a static proxy list, for example, when disabling advertising for mod_cluster. For information about how to disable advertising for mod_cluster, see Disable Advertising for mod_cluster in the JBoss EAP *Configuration Guide*.

For more information about mod_cluster attributes, see ModCluster Subsystem Attributes in the JBoss EAP *Configuration Guide*.

## 4.20. VIEWING CONFIGURATION CHANGES

JBoss EAP 7 provides the ability to track configuration changes made to the running server. This allows administrators to view a history of configuration changes made by authorized users.

In JBoss EAP 7.0, you must use the **core-service** management CLI command to configure options and to list recent configuration changes.

### Example: List Configuration Changes in JBoss EAP 7.0

```
/core-service=management/service=configuration-changes:add(max-history=10)
/core-service=management/service=configuration-changes:list-changes
```

JBoss EAP 7.1 introduced a new **core-management** subsystem that can be configured to track configuration changes made to the running server. This is the preferred method of configuring and viewing configuration changes in JBoss EAP 7.1 and later.

### Example: List Configuration Changes in JBoss EAP 7.1 and Later

```
/subsystem=core-management/service=configuration-changes:add(max-history=20)
/subsystem=core-management/service=configuration-changes:list-changes
```

For more information about using the new **core-management** subsystem introduced in JBoss EAP 7.1, see View Configuration Changes in the JBoss EAP *Configuration Guide*.

# CHAPTER 5. APPLICATION MIGRATION CHANGES

## 5.1. WEB SERVICES APPLICATION CHANGES

JBossWS 5 brings new features and performance improvements to JBoss EAP 7 web services, mainly through upgrades of the Apache CXF, Apache WSS4J, and Apache Santuario components.

### 5.1.1. JAX-RPC Support Changes

The Java API for XML-based RPC (JAX-RPC) was deprecated in Java EE 6 and was optional in Java EE 7. It is no longer available or supported in JBoss EAP 7. Applications that use JAX-RPC must be migrated to use JAX-WS, which is the current Java EE standard web services framework.

Use of JAX-RPC web services can be identified in any of the following ways:

- The presence of a JAX-RPC mapping file, which is an XML file with the root element **<java-wsdl-mapping>**.
- The presence of a **webservices.xml** XML descriptor file that contains a **<webservice-description>** element, which includes a **<jaxrpc-mapping-file>** child element. The following is an example of **webservices.xml** descriptor file that defines a JAX-RPC web service.

```
<webservices xmlns="http://java.sun.com/xml/ns/j2ee"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://www.ibm.com/webservices/xsd/j2ee_web_services_1_1.xsd" version="1.1">
  <webservice-description>
    <webservice-description-name>HelloService</webservice-description-name>
    <wsdl-file>WEB-INF/wsdl/HelloService.wsdl</wsdl-file>
    <jaxrpc-mapping-file>WEB-INF/mapping.xml</jaxrpc-mapping-file>
    <port-component>
      <port-component-name>Hello</port-component-name>
      <wsdl-port>HelloPort</wsdl-port>
      <service-endpoint-interface>org.jboss.chap12.hello.Hello</service-endpoint-interface>
      <service-impl-bean>
        <servlet-link>HelloWorldServlet</servlet-link>
      </service-impl-bean>
    </port-component>
  </webservice-description>
</webservices>
```

- The presence of an **ejb-jar.xml** file, which contains a **<service-ref>** that references a JAX-RPC mapping file.

### 5.1.2. Apache CXF Spring Web Services Changes

In previous releases of JBoss EAP, you could customize the JBossWS and Apache CXF integration by including a **jbossws-cxf.xml** configuration file with the endpoint deployment archive. One use case for this was to configure interceptor chains for web service client and server endpoints on the Apache CXF bus. This integration required Spring to be deployed in the JBoss EAP server.

Spring integration is no longer supported in JBoss EAP 7. Any application that contains a **jbossws-cxf.xml** descriptor configuration file must be modified to replace the custom configuration defined in that file. While it is still possible to directly access the Apache CXF API, be aware that the application will not be portable.

The suggested approach is to replace Spring custom configurations with the new JBossWS descriptor configuration options where possible. The JBossWS descriptor-based approach provides similar functionality without requiring modification of the client endpoint code. In some cases, you can replace Spring with Context Dependency Injection (CDI).

**Apache CXF Interceptors**

The JBossWS descriptor provides new configuration options that allow you to declare the interceptors without modifying the client endpoint code. Instead you declare interceptors within predefined client and endpoint configurations by specifying a list of interceptor class names for the **cxf.interceptors.in** and **cxf.interceptors.out** properties.

The following is an example of a **jaxws-endpoint-config.xml** file that declares interceptors using these properties.

```
<?xml version="1.0" encoding="UTF-8"?>
<jaxws-config xmlns="urn:jboss:jbossws-jaxws-config:4.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:javaee="http://java.sun.com/xml/ns/javaee"
  xsi:schemaLocation="urn:jboss:jbossws-jaxws-config:4.0 schema/jbossws-jaxws-config_4_0.xsd">
  <endpoint-config>
    <config-name>org.jboss.test.ws.jaxws.cxf.interceptors.EndpointImpl</config-name>
    <property>
      <property-name>cxf.interceptors.in</property-name>
      <property-
value>org.jboss.test.ws.jaxws.cxf.interceptors.EndpointInterceptor,org.jboss.test.ws.jaxws.cxf.intercepto
s.FooInterceptor</property-value>
    </property>
    <property>
      <property-name>cxf.interceptors.out</property-name>
      <property-value>org.jboss.test.ws.jaxws.cxf.interceptors.EndpointCounterInterceptor</property-
value>
    </property>
  </endpoint-config>
</jaxws-config>
```

**Apache CXF Features**

The JBossWS descriptor allows you to declare features within predefined client and endpoint configurations by specifying a list of feature class names for the **cxf.features** property.

The following is an example of a **jaxws-endpoint-config.xml** file that declares a feature using this property.

```
<?xml version="1.0" encoding="UTF-8"?>
<jaxws-config xmlns="urn:jboss:jbossws-jaxws-config:4.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:javaee="http://java.sun.com/xml/ns/javaee"
  xsi:schemaLocation="urn:jboss:jbossws-jaxws-config:4.0 schema/jbossws-jaxws-config_4_0.xsd">
  <endpoint-config>
    <config-name>Custom FI Config</config-name>
    <property>
      <property-name>cxf.features</property-name>
      <property-value>org.apache.cxf.feature.FastInfosetFeature</property-value>
    </property>
  </endpoint-config>
</jaxws-config>
```

**Apache CXF HTTP Transport**
In Apache CXF, HTTP transport configuration is achieved by specifying **org.apache.cxf.transport.http.HTTPConduit** options. JBossWS integration allows conduits to be modified programmatically using the Apache CXF API as follows.

```
import org.apache.cxf.frontend.ClientProxy;
import org.apache.cxf.transport.http.HTTPConduit;
import org.apache.cxf.transports.http.configuration.HTTPClientPolicy;

// Set chunking threshold before using a JAX-WS port client
...
HTTPConduit conduit = (HTTPConduit)ClientProxy.getClient(port).getConduit();
HTTPClientPolicy client = conduit.getClient();

client.setChunkingThreshold(8192);
...
```

You can also control and override the Apache CXF **HTTPConduit** default values by setting system properties.

| Property | Type | Description |
|---|---|---|
| cxf.client.allowChunking | Boolean | Specifies whether to send requests using chunking. |
| cxf.client.chunkingThreshold | Integer | Sets the threshold at which switching from non-chunking to chunking mode. |
| cxf.client.connectionTimeout | Long | Sets the number of milliseconds for the connection timeout. |
| cxf.client.receiveTimeout | Long | Sets the number of milliseconds for the receive timeout. |
| cxf.client.connection | String | Specifies whether to use the **Keep-Alive** or **close** connection type. |
| cxf.tls-client.disableCNCheck | Boolean | Specifies whether to disable the CN host name check. |

### 5.1.3. WS-Security Changes

- If your application contains a custom callback handler that accesses the **org.apache.ws.security.WSPasswordCallback** class, be aware that this class has moved to package **org.apache.wss4j.common.ext**.
- Most of the SAML bean objects from the **org.apache.ws.security.saml.ext** package have been moved to the **org.apache.wss4j.common.saml package**.
- Use the RSA v1.5 key transport and all related algorithms are disallowed by default.

- The Security Token Service (STS) previously only validated **onBehalfOf** tokens. It now also validates **ActAs** tokens. As a consequence, a valid username and password must be specified in the **UsernameToken** that is provided for the **ActAs** token.
- SAML Bearer tokens are now required to have an internal signature. The **org.apache.wss4j.dom.validate.SamlAssertionValidator** class now has a **setRequireBearerSignature()** method to enable or disable the signature verification.

### 5.1.4. JBoss Modules Structure Change

The **cxf-api** and **cxf-rt-core** JARs have been merged into one **cxf-core** JAR. As a consequence, the **org.apache.cxf** module in JBoss EAP now contains the **cxf-core** JAR and exposes more classes than in the previous release.

### 5.1.5. Bouncy Castle Requirements and Changes

If you want to use AES encryption with Galois/Counter Mode (GCM) for symmetric encryption in XML/WS-Security, you need the BouncyCastle Security Provider.

JBoss EAP 7 ships with the **org.bouncycastle** module and JBossWS is now able to rely on its class loader to get and use the BouncyCastle Security Provider. Therefore it is no longer necessary to statically install BouncyCastle in the current JVM. For applications running outside of the container, the security provider can be made available to JBossWS by adding a BouncyCastle library to the class path.

You can disable this behavior by setting the **org.jboss.ws.cxf.noLocalBC** property value to **true** in the **jaxws-endpoint-config.xml** deployment descriptor file for the server or the **jaxws-client-config.xml** descriptor file for clients.

If you want to use a different version than the one that ships with JBoss EAP, you can still statically install BouncyCastle to the JVM. In that case, the statically installed BouncyCastle Security Provider is chosen over the provider present in the class path. To avoid any issues, you must use BouncyCastle 1.49, 1.51, or greater.

### 5.1.6. Apache CXF Bus Selection Strategy

The default bus selection strategy for clients running in-container has changed from **THREAD_BUS** to **TCCL_BUS**. For clients running out-of container, the default strategy is still **THREAD_BUS**. You can restore the behavior to that of the previous release by using either of the following methods.

- Boot the JBoss EAP server with the system property **org.jboss.ws.cxf.jaxws-client.bus.strategy** value set to **THREAD_BUS**.
- Explicitly set the selection strategy in the client code.

### 5.1.7. JAX-WS 2.2 Requirements for WebServiceRef

Containers must use JAX-WS 2.2 style constructors, which include the WebServiceFeature class as an argument in the constructor, to build clients that are injected into web service references. JBoss EAP 6.4, which ships with JBossWS 4, hides that requirement. JBoss EAP 7 ships with JBossWS 5, which no longer hides this requirement. This means that user provided service classes injected by the container must implement JAX-WS 2.2 or later by updating the existing code to use the **javax.xml.ws.Service** constructor that includes one or more **WebServiceFeature** arguments.

```
protected Service(URL wsdlDocumentLocation,
        QName serviceName,
        WebServiceFeature... features)
```

### 5.1.8. IgnoreHttpsHost CN Check Change

In previous releases, you could disable the HTTPS URL hostname check against a service's Common Name (CN) given in its certificate by setting the system property **org.jboss.security.ignoreHttpsHost** to **true**. This system property name has been replaced with **cxf.tls-client.disableCNCheck**.

### 5.1.9. Server Side Configuration and Class Loading

As a consequence of enabling injections into service endpoint and service client handlers, it is no longer possible to automatically load handler classes from the **org.jboss.as.webservices.server.integration** JBoss module. If your application depends on a given predefined configuration, you might need to explicitly define new module dependencies for your deployment. For more information, see Migrate Explicit Module Dependencies

### 5.1.10. Deprecation of Java Endorsed Standards Override Mechanism

The Java Endorsed Standards Override Mechanism was deprecated in JDK 1.8_40 with intent to remove it in JDK 9. This mechanism allowed developers to make libraries available to all deployed applications by placing JARs into an endorsed directory within the JRE.

If your application uses the JBossWS implementation of Apache CXF, JBoss EAP 7 ensures the required dependencies are added in the correct order and you should not be impacted by this change. If your application accesses Apache CXF directly, you must now provide the Apache CXF dependencies after the JBossWS dependencies as part of your application deployment.

### 5.1.11. Specification of Descriptor in EAR Archive

In previous releases of JBoss EAP, you could configure the **jboss-webservices.xml** deployment descriptor file for EJB web service deployments in the **META-INF/** directory of JAR archives or in the **WEB-INF/** directory for POJO web service deployments and EJB web service endpoints bundled in WAR archives.

In JBoss EAP 7, you can now configure the **jboss-webservices.xml** deployment descriptor file in the **META-INF/** directory of an EAR archive. If a **jboss-webservices.xml** file is found both in the EAR archive and the JAR or WAR archive, the configuration data in the **jboss-webservices.xml** file for the JAR or WAR overrides the corresponding data in the EAR descriptor file.

## 5.2. UPDATE THE REMOTE URL CONNECTOR AND PORT

In JBoss EAP 7, the default connector has changed from **remote** to **http-remoting** and the default remote connection port has changed from **4447** to **8080**. The JNDI provider URL for the default configuration has changed from **remote://localhost:4447** to **http-remoting://localhost:8080**.

If you use the JBoss EAP 7 **migrate** operation to update your configuration, you do not need to modify the remote connector, remote port, or JNDI provider URLs because the migration operation preserves the JBoss EAP 6 remoting connector and **4447** port configuration settings in the subsystem configuration. For more information about the **migrate** operation, see Management CLI Migration Operation.

If you do not use the **migrate** operation and instead run with the new JBoss EAP 7 default configuration, you must change the remote connector, remote port, and JNDI provider URL to use the new settings as described above.

## 5.3. MESSAGING APPLICATION CHANGES

### 5.3.1. Replace or Update JMS Deployment Descriptors

The proprietary HornetQ JMS resource deployment descriptor files identified by the naming pattern **-jms.xml** no longer work in JBoss EAP 7. The following is an example of a JMS resource deployment descriptor file in JBoss EAP 6.

```
<?xml version="1.0" encoding="UTF-8"?>
<messaging-deployment xmlns="urn:jboss:messaging-deployment:1.0">
  <hornetq-server>
    <jms-destinations>
      <jms-queue name="testQueue">
        <entry name="queue/test"/>
        <entry name="java:jboss/exported/jms/queue/test"/>
      </jms-queue>
      <jms-topic name="testTopic">
        <entry name="topic/test"/>
        <entry name="java:jboss/exported/jms/topic/test"/>
      </jms-topic>
    </jms-destinations>
  </hornetq-server>
</messaging-deployment>
```

If you used **-jms.xml** JMS deployment descriptors in your application in the previous release, you can either convert your application to use the standard Java EE deployment descriptor as specified in section EE.5.18 of the Java EE 7 specification or you can update the deployment descriptor to use the **messaging-activemq-deployment** schema instead.

If you choose to update the descriptor, you need to make the following modifications.

- Change the namespace from "urn:jboss:messaging-deployment:1.0" to "urn:jboss:messaging-activemq-deployment:1.0".
- Change the **<hornetq-server>** element name to **<server>**.

The modified file should look like the following example.

```
<?xml version="1.0" encoding="UTF-8"?>
<messaging-deployment xmlns="urn:jboss:messaging-activemq-deployment:1.0">
  <server>
    <jms-destinations>
      <jms-queue name="testQueue">
        <entry name="queue/test"/>
        <entry name="java:jboss/exported/jms/queue/test"/>
      </jms-queue>
      <jms-topic name="testTopic">
        <entry name="topic/test"/>
        <entry name="java:jboss/exported/jms/topic/test"/>
      </jms-topic>
      </jms-destinations>
    </server>
  </messaging-deployment>
```

For information about server configuration changes related to messaging, see Messaging Server Configuration Changes.

### 5.3.2. Update External JMS Clients

JBoss EAP 7 still supports the JMS 1.1 API, so you do not need to modify your code.

The default remote connector and port has changed in JBoss EAP 7. For details about this change, see Update the Remote URL Connector and Port .

If you migrate your server configuration using the **migrate** operation, the old settings are preserved and you do not need to update your **PROVIDER_URL**. However, if you run with the new JBoss EAP 7 default configuration, you must change the **PROVIDER_URL** in the client code to use the new **http-remoting://localhost:8080** setting. For more information, see Migrate Remote Naming Client Code.

If you plan to migrate your code to use the JMS 2.0 API, see the **helloworld-jms** quickstart for a working example.

### 5.3.3. Replace the HornetQ API

JBoss EAP 6 included the **org.hornetq** module, which allowed you to use the HornetQ API in your application source code.

Apache ActiveMQ Artemis replaces HornetQ in JBoss EAP 7, so you must migrate any code that used the HornetQ API to use the Apache ActiveMQ Artemis API. The libraries for this API are included in the **org.apache.activemq.artemis** module.

ActiveMQ Artemis is an evolution of HornetQ, so many of the concepts still apply.

### 5.3.4. Replace Deprecated Address Setting Attributes

The ability to auto-create and auto-delete topics and queues using the **auto-create-jms-queues**, **auto-delete-jms-queues**, **auto-create-jms-topics**, and **auto-delete-jms-topics** attributes was only partially implemented and not fully configurable in JBoss EAP 7. These attributes, which are deprecated, were provided as a technology preview feature only and were not supported.

You must replace any usage of these deprecated attributes with the following replacement attributes.

NOTE

The deprecated attributes no longer configure this functionality in JBoss EAP 7.2 and do not take effect. The replacement attributes are not supported either. They are provided only as a way to migrate on the best effort basis.

| Deprecated Attribute | Replacement Attribute |
|---|---|
| auto-create-jms-queues | auto-create-queues |
| auto-delete-jms-queues | auto-delete-queues |

| Deprecated Attribute | Replacement Attribute |
|---|---|
| auto-create-jms-topics | auto-create-addresses |
| auto-delete-jms-topics | auto-delete-addresses |

For more information about these replacement attributes, see Address Setting Attributes in the *Configuring Messaging*.

### 5.3.5. Messaging Application Changes Required for JBoss EAP 7.2

Starting with JBoss EAP 7.2, if a client application directly depends on Artemis client JARs, for example, **artemis-jms-client**, **artemis-commons**, **artemis-core-client**, or **artemis-selector**, then you must add the following dependency in your **pom.xml** file for **wildfly-client-properties**.

```
<dependency>
  <groupId>org.jboss.eap</groupId>
  <artifactId>wildfly-client-properties</artifactId>
</dependency>
```

This is to avoid a **JMSRuntimeException** when calling **message.getJMSReplyTo()** from an older JBoss EAP 7 client as described in JBEAP-15889.

## 5.4. JAX-RS AND RESTEASY APPLICATION CHANGES

JBoss EAP 6 bundled RESTEasy 2, which was an implementation of JAX-RS 1.x.

JBoss EAP 7.0 and JBoss EAP 7.1 included RESTEasy 3.0.x, which is an implementation of JAX-RS 2.0 as defined by the JSR 339: JAX-RS 2.0: The Java API for RESTful Web Services specification. For more information about the Java API for RESTful Web Services, see the JAX-RS 2.0 API Specification.

JBoss EAP 7.2 includes RESTEasy 3.6.1, which is an implementation of JAX-RS 2.1 as defined by the JSR 370: Java(TM )API for RESTful Web Services (JAX-RS 2.1) Specification. This release also adds support for JDK 11. While providing some of the RESTEasy 4 key features, this release is based on RESTEasy 3.0, ensuring full backward compatibility. As a result, you should encounter few issues when migrating from RESTEasy 3.0.x to 3.6.1. For more information about the Java API for RESTEasy 3.6.1, see RESTEasy JAX-RS 3.6.1.Final API.

If you are migrating from JBoss EAP 6.4, be aware that the version of Jackson included in JBoss EAP has changed. JBoss EAP 6.4 included Jackson 1.9.9. JBoss EAP 7 and later now include Jackson 2.6.3 or greater.

This section describes how these changes might impact applications that use RESTEasy or JAX-RS.

### 5.4.1. RESTEasy Deprecated Classes

**Interceptor and MessageBody Classes**
JSR 311: JAX-RS: The Java™ API for RESTful Web Services did not include an interceptor framework, so RESTEasy 2 provided one. JSR 339: JAX-RS 2.0: The Java API for RESTful Web Services introduced an official interceptor and filter framework, so the interceptor framework included in RESTEasy 2 is now deprecated, and was replaced by the JAX-RS compliant interceptor facility in RESTEasy 3.x. The relevant interfaces are defined in the **javax.ws.rs.ext** package of the **jaxrs-api** module.

- The following interceptor interfaces are deprecated in RESTEasy 3.x.
  - **org.jboss.resteasy.spi.interception.PreProcessInterceptor**
  - **org.jboss.resteasy.spi.interception.PostProcessInterceptor**
  - **org.jboss.resteasy.spi.interception.ClientExecutionInterceptor**
  - **org.jboss.resteasy.spi.interception.ClientExecutionContext**
  - **org.jboss.resteasy.spi.interception.AcceptedByMethod**
- The **org.jboss.resteasy.spi.interception.PreProcessInterceptor** interface was replaced by the **javax.ws.rs.container.ContainerRequestFilter** interface in RESTEasy 3.x.
- The following interfaces and classes are also deprecated in RESTEasy 3.x.
  - **org.jboss.resteasy.spi.interception.MessageBodyReaderInterceptor**
  - **org.jboss.resteasy.spi.interception.MessageBodyWriterInterceptor**
  - **org.jboss.resteasy.spi.interception.MessageBodyWriterContext**
  - **org.jboss.resteasy.spi.interception.MessageBodyReaderContext**
  - **org.jboss.resteasy.core.interception.InterceptorRegistry**
  - **org.jboss.resteasy.core.interception.InterceptorRegistryListener**
  - **org.jboss.resteasy.core.interception.ClientExecutionContextImpl**
- The **org.jboss.resteasy.spi.interception.MessageBodyWriterInterceptor** interface was replaced by the **javax.ws.rs.ext.WriterInterceptor** interface.
- In addition, some changes to the **javax.ws.rs.ext.MessageBodyWriter** interface might not be backward compatible with respect to JAX-RS 1.x. If your application used JAX-RS 1.x, review your application code to make sure you define **@Produces** or **@Consumes** for your endpoints. Failure to do so might result in an error similar to the following.

```
org.jboss.resteasy.core.NoMessageBodyWriterFoundFailure: Could not find
MessageBodyWriter for response object of type: <OBJECT> of media type:
```

  The following is an example of a REST endpoint that can cause this error.

```
@Path("dates")
public class DateService {

    @GET
    @Path("daysuntil/{targetdate}")
    public long showDaysUntil(@PathParam("targetdate") String targetDate) {
        DateLogger.LOGGER.logDaysUntilRequest(targetDate);
        final long days;

        try {
            final LocalDate date = LocalDate.parse(targetDate, DateTimeFormatter.ISO_DATE);
            days = ChronoUnit.DAYS.between(LocalDate.now(), date);
        } catch (DateTimeParseException ex) {
            // ** DISCLAIMER **. This example is contrived.
            throw new
WebApplicationException(Response.status(400).entity(ex.getLocalizedMessage()).type(Media
Type.TEXT_PLAIN)
                .build());
        }
        return days;
    }
}
```

To fix the issue, add the import for **javax.ws.rs.Produces** and the **@Produces** annotation as follows.

```
...
import javax.ws.rs.Produces;
...

@Path("dates")
public class DateService {

    @GET
    @Path("daysuntil/{targetdate}")
    @Produces(MediaType.TEXT_PLAIN)
    public long showDaysUntil(@PathParam("targetdate") String targetDate) {
        DateLogger.LOGGER.logDaysUntilRequest(targetDate);
        final long days;

        try {
            final LocalDate date = LocalDate.parse(targetDate, DateTimeFormatter.ISO_DATE);
            days = ChronoUnit.DAYS.between(LocalDate.now(), date);
        } catch (DateTimeParseException ex) {
          // ** DISCLAIMER **. This example is contrived.
          throw new
WebApplicationException(Response.status(400).entity(ex.getLocalizedMessage()).type(Media
Type.TEXT_PLAIN)
                .build());
        }
        return days;
    }
}
```

NOTE

All interceptors from the previous release of RESTEasy can run in parallel with the new JAX-RS filter and interceptor interfaces.

For more information about interceptors, see RESTEasy Interceptors in *Developing Web Services Applications* for JBoss EAP.

For more information about the new replacement API, see the RESTEasy JAX-RS 3.6.1.Final API.

**Client API**

The RESTEasy client framework in **resteasy-jaxrs** was replaced by the JAX-RS 2.0 compliant **resteasy-client** module in JBoss EAP 7.0. As a result, some RESTEasy client API classes and methods are deprecated.

- The following classes are deprecated.
  - **org.jboss.resteasy.client.ClientRequest**
  - **org.jboss.resteasy.client.ClientRequestFactory**
  - **org.jboss.resteasy.client.ClientResponse**
  - **org.jboss.resteasy.client.ProxyBuilder**
  - **org.jboss.resteasy.client.ProxyConfig**
  - **org.jboss.resteasy.client.ProxyFactory**
- The **org.jboss.resteasy.client.ClientResponseFailure** exception and the **org.jboss.resteasy.client.ClientExecutor** and **org.jboss.resteasy.client.EntityTypeFactory** interfaces are also deprecated.
- You must replace the **org.jboss.resteasy.client.ClientRequest** and **org.jboss.resteasy.client.ClientResponse** classes with **org.jboss.resteasy.client.jaxrs.ResteasyClient** and **javax.ws.rs.core.Response** respectively. The following is an example of how to send a link header with the RESTEasy client in RESTEasy 2.3.x.

  ```
  ClientRequest request = new ClientRequest(generateURL("/linkheader/str"));
  request.addLink("previous chapter", "previous", "http://example.com/TheBook/chapter2",
  null);
  ClientResponse response = request.post();
  LinkHeader header = response.getLinkHeader();
  ```

  The following is an example of how to accomplish the same task with the RESTEasy client in RESTEasy 3.

  ```
  ResteasyClient client = new ResteasyClientBuilder().build();
  Response response = client.target(generateURL("/linkheader/str")).request()
    .header("Link", "<http://example.com/TheBook/chapter2>; rel=\"previous\";
  title=\"previous chapter\"").post(Entity.text(new String()));
  javax.ws.rs.core.Link link = response.getLink("previous");
  ```

  See the **resteasy-jaxrs-client** quickstart for an example of an external JAX-RS RESTEasy client that interacts with a JAX-RS Web service.
- The classes and interfaces in the **org.jboss.resteasy.client.cache** package are also deprecated. They are replaced by equivalent classes and interfaces in the **org.jboss.resteasy.client.jaxrs.cache** package.

NOTE

For more information about the **org.jboss.resteasy.client.jaxrs** API classes, see the RESTEasy JAX-RS JavaDoc.

**StringConverter**

The **org.jboss.resteasy.spi.StringConverter** class is deprecated in RESTEasy 3.x. This functionality can be replaced using the JAX-RS jax.ws.rs.ext.ParamConverterProvider class.

### 5.4.2. Removed or Protected RESTEasy Classes

#### ResteasyProviderFactory Add methods

Most of the **org.jboss.resteasy.spi.ResteasyProviderFactory** **add()** methods have been removed or made protected in RESTEasy 3.0. For example, the **addBuiltInMessageBodyReader()** and **addBuiltInMessageBodyWriter()** methods have been removed and the **addMessageBodyReader()** and **addMessageBodyWriter()** methods have been made protected.

You should now use the **registerProvider()** and **registerProviderInstance()** methods.

#### Additional Classes Removed From RESTEasy 3

The **@org.jboss.resteasy.annotations.cache.ServerCached** annotation, which specified the response to the JAX-RS method should be cached on the server, was removed from RESTEasy 3 and must be removed from the application code.

### 5.4.3. Additional RESTEasy Changes

#### SignedInput and SignedOuput

- **SignedInput** and **SignedOutput** for **resteasy-crypto** must have the **Content-Type** set to **multipart/signed** in either the **Request** or **Response** object, or by using the **@Consumes** or **@Produces** annotation.
- **SignedOutput** and **SignedInput** can be used to return the **application/pkcs7-signature** MIME type format in binary form by setting that type in the **@Produces** or **@Consumes** annotations.
- If the **@Produces** or **@Consumes** is **text/plain** MIME type, **SignedOutput** will be base64 encoded and sent as a String.

#### Security Filters

The security filters for **@RolesAllowed**, **@PermitAll**, and **@DenyAll** now return "403 Forbidden" instead of "401 Unauthorized".

#### Client-side Filters

The client-side filters that were introduced in JAX-RS 2.0 will not be bound and run when you are using the RESTEasy client API from a release prior to RESTEasy 3.0.

#### Asynchronous HTTP Support

Because the JAX-RS 2.0 specification added asynchronous HTTP support using the **@Suspended** annotation and the **AsynResponse** interface, the RESTEasy proprietary API for asynchronous HTTP was deprecated and might be removed in a future RESTEasy release. The asynchronous Tomcat and asynchronous JBoss Web modules have also been removed from the server installation. If you are not using the Servlet 3.0 container or higher, asynchronous HTTP server-side processing will be simulated and run synchronously in same request thread.

#### Server-side Cache

Server-side cache setup has changed. Please see the RESTEasy Documentation for more information.

#### YAML Provider Setting Changes

In previous releases of JBoss EAP, the RESTEasy YAML provider setting was enabled by default. This has changed in JBoss EAP 7. The YAML provider is now disabled by default. Its use is not supported due to a security issue in the **SnakeYAML** library used by RESTEasy for unmarshalling and it must be explicitly enabled in the application. For information about how to enable the YAML provider in your application and add the Maven dependencies, see YAML Provider in *Developing Web Services Applications* for JBoss EAP.

#### Default Charset UTF-8 in Content-Type Header

As of JBoss EAP 7.1, the **resteasy.add.charset** parameter is set to **true** by default. You can set the **resteasy.add.charset** parameter to **false** if you do not want RESTEasy to add **charset=UTF-8** to the returned content-type header when the resource method returns a **text/*** or **application/xml*** media type without an explicit charset.

For more information about text media types and character sets, see Text Media Types and Character Sets in *Developing Web Services Applications* for JBoss EAP.

**SerializableProvider**

Deserializing Java objects from untrusted sources is not safe. For this reason, in JBoss EAP 7, the **org.jboss.resteasy.plugins.providers.SerializableProvider** class is disabled by default, and it is not recommended to use this provider.

**Matching Requests to Resource Methods**

In RESTEasy 3, improvements and corrections were made to the implementation of matching rules, as defined in the JAX-RS specification. In particular, a change was made to how an ambiguous URI on a sub-resource method and a sub-resource locator is handled.

In RESTEasy 2, it was possible for a sub-resource locator to execute successfully even when there was another sub-resource with the same URI. This behavior was incorrect according to the specification.

In RESTEasy 3, when there is an ambiguous URI for a sub-resource and a sub-resource locator, calling the sub-resource will be successful; however, calling the sub-resource locator will result in an HTTP status **405 Method Not Allowed** error.

The following example contains an ambiguous **@Path** annotation on a sub-resource method and a sub-resource locator. Notice that the URI to both endpoints, **anotherResource** and **anotherResourceLocator**, is the same. The difference between the two endpoints is that the **anotherResource** method is associated with the REST verb, **POST**. The **anotherResourceLocator** method is not associated with any REST verb. According to the specification, the endpoint with the REST verb, in this case the **anotherResource** method, will always be selected.

```
@Path("myResource")
public class ExampleSubResources {
   @POST
   @Path("items")
   @Produces("text/plain")
   public Response anotherResource(String text) {
      return Response.ok("ok").build();
   }

   @Path("items")
   @Produces("text/plain")
   public SubResource anotherResourceLocator() {
      return new SubResource();
   }
}
```

**Resource Method Algorithm Switch**

A bug discovered in the resource method matching algorithm used in RESTEasy 3.0.x versions prior to 3.0.25.Final caused RESTEasy to return too many resource methods when responding to requests.

There are three stages in the matching algorithm:

1. Use the request path to choose possible resource classes.

2. Use the request path to choose possible resource methods.

3. Use the HTTP verb and media types, coming and going, to choose a final resource method.

According to the JAX-RS 2.0 specification, after the set of potential resource methods is sorted, only the maximal elements should be passed on to step 3. However, RESTEasy 3.0.x implementations prior to RESTEasy 3.0.25 passed *all* methods to step 3. RESTEasy 3.0.24, which was included in JBoss EAP 7.1.0, exhibits this incorrect behavior.

RESTEasy 3.0.25, which is included in JBoss EAP 7.1.1, provides the fix to limit the methods passed to step 3 to be compliant with the JAX-RS 2.0 specification. Because the looser behavior might be preferable, RESTEasy 3.0.25 also introduces a **context-param** configuration option, **resteasy.loose.step2.request.matching**, which defaults to **false**, that can be configured to enable the old behavior.

If you update your JBoss EAP server from 7.1.0 to 7.1.1 and you want to keep the old behavior and pass all potential resource methods to step 3, set the **resteasy.loose.step2.request.matching** option to **true**.

The matching algorithm was changed in the JAX-RS 2.1 specification to pass all matching resource methods to step 3. RESTEasy 3.6.1, which is included in JBoss EAP 7.2, provides a **jaxrs.2.0.request.matching** option to to retain the stricter behavior as defined in the JAX-RS 2.0 specification.

If you migrate your application from JBoss EAP from 7.1.0 to 7.2.x, you should not see a change in the behavior of the resource method matching algorithm. If you migrate your application from JBoss EAP from 7.1.1 to 7.2.x and want to retain the stricter behavior as defined in the JAX-RS 2.0 specification, set the **jaxrs.2.0.request.matching** option to **true**.

### 5.4.4. RESTEasy SPI Changes

#### SPI Exceptions

All SPI failure exceptions were deprecated and are no longer used internally. They have been replaced with the corresponding JAX-RS exception.

| Deprecated Exception | Replacement Exception in **jaxrs-api** module |
|---|---|
| org.jboss.resteasy.spi.ForbiddenException | javax.ws.rs.ForbiddenException |
| org.jboss.resteasy.spi.MethodNotAllowedException | javax.ws.rs.NotAllowedException |
| org.jboss.resteasy.spi.NotAcceptableException | javax.ws.rs.NotAcceptableException |
| org.jboss.resteasy.spi.NotFoundException | javax.ws.rs.NotFoundException |
| org.jboss.resteasy.spi.UnauthorizedException | javax.ws.rs.NotAuthorizedException |
| org.jboss.resteasy.spi.UnsupportedMediaTypeException | javax.ws.rs.NotSupportedException |

#### InjectorFactory and Registry

The **InjectorFactory** and **Registry** SPIs have changed. This should not be an issue if you use RESTEasy as documented and supported.

### 5.4.5. Jackson Provider Changes

The version of Jackson included in JBoss EAP has changed. The previous version of JBoss EAP included Jackson 1.9.9. JBoss EAP 7 now includes Jackson 2.6.3 or greater. As a result, the Jackson provider has changed from **resteasy-jackson-provider** to **resteasy-jackson2-provider**.

The upgrade to the **resteasy-jackson2-provider** requires some package changes. For example, the Jackson annotation package has changed from **org.codehaus.jackson.annotate** to **com.fasterxml.jackson.annotation**.

To switch your application to use the default provider that was included in the previous release of JBoss EAP, see Switching the Default Jackson Provider in *Developing Web Services Applications* for JBoss EAP.

### 5.4.6. Spring RESTEasy Integration Changes

The Spring 4.0 framework introduced support for Java 8. If you plan to use the RESTEasy 3.x integration with Spring, be sure to specify 4.2.x as the minimum Spring version in your deployment as this is the earliest stable version supported by JBoss EAP 7.

### 5.4.7. RESTEasy Jettison JSON Provider Changes

The RESTEasy Jettison JSON provider is deprecated in JBoss EAP 7 and is no longer added to deployments by default. You are encouraged to switch to the recommended RESTEasy Jackson provider. If you prefer to continue to use the Jettison provider, you must define an explicit dependency for it in the **jboss-deployment-descriptor.xml** file as demonstrated in the following example.

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss-deployment-structure>
  <deployment>
    <exclusions>
      <module name="org.jboss.resteasy.resteasy-jackson2-provider"/>
      <module name="org.jboss.resteasy.resteasy-jackson-provider"/>
    </exclusions>
    <dependencies>
      <module name="org.jboss.resteasy.resteasy-jettison-provider" services="import"/>
    </dependencies>
  </deployment>
</jboss-deployment-structure>
```

For more information about how to define explicit dependencies, see Add an Explicit Module Dependency to a Deployment in the JBoss EAP *Development Guide*.

## 5.5. CDI APPLICATION CHANGES

JBoss EAP 7.2 includes support for CDI 2.0. As a result, applications written using CDI 1.0 or CDI 1.2 might see some changes in behavior when migrating to JBoss EAP 7.2. This section summarizes only few of the changes made in CDI 1.2 and CDI 2.0.

You can find more information about Weld and CDI 2.0 in the following references:

- JSR 365: Contexts and Dependency Injection for Java 2.0
- CDI 2.0 Javadoc

- Weld 3.0.5.Final - CDI Reference Implementation

### Bean Archives

Bean classes of enabled beans must be deployed in bean archives to ensure they are scanned by CDI to find and process the bean classes.

In CDI 1.0, an archive was defined as an *explicit* bean archive if it contained a **beans.xml** file in the **META-INF/** directory for an application client, EJB, or library JAR, or if it contained a **beans.xml** file in the **WEB-INF/** directory for a WAR.

CDI 1.1 introduced *implicit* bean archives, which are archives that contain one or more bean classes with a bean defining annotation, or one or more session beans. Implicit bean archives are scanned by CDI and, during type discovery, only classes with bean defining annotations are discovered. For more information, see Type and Bean Discovery in *JSR 365: Contexts and Dependency Injection for Java 2.0* .

A bean archive has a bean discovery mode of **all**, **annotated** or **none**. A bean archive that contains a **beans.xml** file with no version has a default bean discovery mode of **all**. A bean archive that contains a **beans.xml** file with version **1.1** or later must specify the **bean-discovery-mode** attribute. The default value for the attribute is **annotated**.

An archive is not a bean archive in the following cases:

- It contains a **beans.xml** file with a **bean-discovery-mode** of **none**.
- It contains a CDI extension with no **beans.xml** file.

An archive is an *explicit* bean archive in the following cases:

- The archive contains a **beans.xml** file with a version number of 1.1 or later and a **bean-discovery-mode** of **all**.
- The archive contains a **beans.xml** file with no version number.
- The archive contains an empty **beans.xml** file.

An archive is an *implicit* bean archive in the following cases:

- The archive contains one or more bean classes with a bean defining annotation, or one or more session beans, even if it does not contain a **beans.xml** file.
- The archive contains a **beans.xml** file with a **bean-discovery-mode** of **annotated**.

CDI 1.2 limited bean defining annotations to the following:

- **@ApplicationScoped**, **@SessionScoped**, **@ConversationScoped**, and **@RequestScoped** annotations
- All other normal scope types
- **@Interceptor** and **@Decorator** annotations
- All stereotype annotations, which are annotations annotated with **@Stereotype**
- **@Dependent** scope annotation

For more information about bean archives, see Bean Archives in *JSR 365: Contexts and Dependency Injection for Java 2.0*.

### Clarification of Conversation Resolution

The conversation context lifecycle was changed in CDI 1.2 to prevent conflicts with the Servlet specification as described in CDI Specification Issue CDI-411 . The conversation scope is active during all servlet requests and should not prevent other servlets or servlet filters from setting the request body or character encoding. For more information, see Conversation context lifecycle in Java EE in *JSR 365: Contexts and Dependency Injection for Java 2.0*.

### Observer Resolution

Event resolution was partly rewritten in CDI 1.2. In CDI 1.0, an event is delivered to an observer method if the observer method has all the event qualifiers. In CDI 1.2, an event is delivered to an observer method if the observer method has no event qualifiers or has a subset of the event qualifiers. For more information, see Observer resolution in *JSR 365: Contexts and Dependency Injection for Java 2.0* .

## 5.6. HTTP SESSION ID CHANGE

The string returned by the **request.getSession().getId()** call to get the unique identifier assigned to an HTTP session has changed between JBoss EAP 6.4 and JBoss EAP 7.

JBoss EAP 6.4 returned both the session ID and the instance ID in the **session-id.instance-id** format.

JBoss EAP 7 returns only the session ID.

This change can create issues with routeless cookies for some upgrades from JBoss EAP 6 to JBoss EAP 7. If your application recreates JSESSIONID cookies based upon the return value from this method call, you might need to update the application code to provide the desired behavior.

## 5.7. MIGRATE EXPLICIT MODULE DEPENDENCIES

The introduction of the modular class loading system and JBoss Modules in the previous release of JBoss EAP allowed for fine-grained control of the classes available to applications. This feature allowed you to configure explicit module dependencies using the application's **MANIFEST.MF** file or the **jboss-deployment-structure.xml** deployment descriptor file.

If you defined explicit module dependencies in your application, you should be aware of the following changes in JBoss EAP 7.

### Review Dependencies for Availability

The modules that are included in JBoss EAP have changed. When you migrate your application to JBoss EAP 7, review your **MANIFEST.MF** and **jboss-deployment-structure.xml** file entries to make sure they do not refer to any modules that were removed from this release of the product.

### Dependencies That Require Annotation Scanning

In the previous release of JBoss EAP, if your dependency contained annotations that needed to be processed during annotation scanning, such as when declaring EJB Interceptors, you were required to generate and include a Jandex index in a new JAR file and then set a flag in the **MANIFEST.MF** or **jboss-deployment-structure.xml** deployment descriptor file.

JBoss EAP 7 now provides automatic runtime generation of annotation indexes for static modules, so you no longer need to generate them manually. However, you still need to add the **annotations** flag to the application's **MANIFEST.MF** file or the **jboss-deployment-structure.xml** deployment descriptor file as demonstrated below.

#### Example: Annotation Flag in the MANIFEST.MF File

```
Dependencies: com.company.my-ejb annotations, com.company.other
```

**Example: Annotation Flag in the jboss-deployment-structure.xml File**

```
<jboss-deployment-structure>
  <deployment>
    <dependencies>
      <module name="com.company.my-ejb" annotations="true"/>
      <module name="com.company.other"/>
    </dependencies>
  </deployment>
</jboss-deployment-structure>
```

## 5.8. HIBERNATE AND JPA MIGRATION CHANGES

### 5.8.1. Hibernate ORM 3.0

The integration classes that made it easier to use Hibernate ORM 3 in JBoss EAP 6.4 were removed from JBoss EAP 7. If your application still uses Hibernate ORM 3 libraries, it is strongly recommended that you migrate your application to use Hibernate ORM 5 as Hibernate ORM 3 will no longer work in JBoss EAP without a lot of effort. If you can not migrate to Hibernate ORM 5, you must define a custom JBoss Module for the Hibernate ORM 3 JARs and exclude the Hibernate ORM 5 classes from your application.

### 5.8.2. Hibernate ORM 4.0 - 4.3

If your application needs second-level cache enabled, be aware that Infinispan 8.x was integrated with Hibernate ORM 5.0. Support for using Infinispan as a Hibernate 2nd-level cache provider was then moved to the Infinispan project in Hibernate ORM 5.3, and as a result, the **hibernate-infinispan** module was dropped from that release.

Applications written with Hibernate ORM 4.x can still use Hibernate ORM 4.x. You must define a custom JBoss module for the Hibernate ORM 4.x JARs and exclude the Hibernate ORM 5 classes from your application. However, it is strongly recommended that you rewrite your application code to use Hibernate ORM 5. For information about migrating to Hibernate ORM 5, see Migrating to Hibernate ORM 5.

### 5.8.3. Migrating to Hibernate ORM 5

JBoss EAP 7.0 included Hibernate ORM 5.0. This section highlights the changes you need to make when migrating from Hibernate ORM version 4.3 to version 5. For more information about the changes implemented between Hibernate ORM 4 and Hibernate ORM 5, see the Hibernate ORM 5.0 Migration Guide.

**Removed and Deprecated Classes**
The following deprecated classes were removed from Hibernate ORM 5:

- **org.hibernate.cfg.AnnotationConfiguration**
- **org.hibernate.id.TableGenerator**
- **org.hibernate.id.TableHiLoGenerator**
- **org.hibernate.id.SequenceGenerator**

**Other Changes to Classes and Packages**

- The **org.hibernate.integrator.spi.Integrator** interface changed to account for bootstrap redesign.
- A new package **org.hibernate.engine.jdbc.env.spi** was created. It contains the **org.hibernate.engine.jdbc.env.spi.JdbcEnvironment** interface, which was extracted from the **org.hibernate.engine.jdbc.spi.JdbcServices** interface.
- A new **org.hibernate.boot.model.relational.ExportableProducer** interface was introduced that will affect **org.hibernate.id.PersistentIdentifierGenerator** implementations.
- The signature of **org.hibernate.id.Configurable** was changed to accept **org.hibernate.service.ServiceRegistry** rather than just **org.hibernate.dialect.Dialect**.
- The **org.hibernate.metamodel.spi.TypeContributor** interface has migrated to **org.hibernate.boot.model.TypeContributor**.
- The **org.hibernate.metamodel.spi.TypeContributions** interface has migrated to **org.hibernate.boot.model.TypeContributions**.

### Type Handling

- Built-in **org.hibernate.type.descriptor.sql.SqlTypeDescriptor** implementations no longer auto-register themselves with **org.hibernate.type.descriptor.sql.SqlTypeDescriptorRegistry**. Applications using custom **SqlTypeDescriptor** implementations that extend the built-in implementations and rely on that behavior must be updated to call **SqlTypeDescriptorRegistry.addDescriptor()** themselves.
- For IDs defined as generated UUIDs, some databases require you to explicitly set the **@Column(length=16)** in order to generate **BINARY(16)** so that comparisons work properly.
- For **EnumType** mappings defined in the **hbm.xml**, where you want **javax.persistence.EnumType.STRING** **name-mapping**, this configuration must be explicitly stated by using either the **useNamed(true)** setting or by specifying a VARCHAR value of **12**.

### Transaction Management

- The transaction SPI underwent a major redesign in Hibernate ORM 5. In Hibernate ORM 4.3, you used the **org.hibernate.Transaction** API to directly access different back-end transaction strategies. Hibernate ORM 5 introduced a level of indirection. On the back end, the **org.hibernate.Transaction** implementation now talks to a **org.hibernate.resource.transaction.TransactionCoordinator**, which represents the transactional context for a given session according to the back-end strategy. While this does not have a direct impact on developers, it could affect the bootstrap configuration. Previously applications would specify **hibernate.transaction.factory_class** property, which is now deprecated, and refer to a **org.hibernate.engine.transaction.spi.TransactionFactory** FQN (fully qualified name). With Hibernate ORM 5, you specify the **hibernate.transaction.coordinator_class** setting and refer to a **org.hibernate.resource.transaction.TransactionCoordinatorBuilder**. See **org.hibernate.cfg.AvailableSettings.TRANSACTION_COORDINATOR_STRATEGY** for additional details.
- The following short names are now recognized:
  - **jdbc**: Manage transactions using the JDBC **java.sql.Connection**. This is the default for non-JPA transactions.
  - **jta**: Manage transactions using JTA.

IMPORTANT

If a JPA application does not provide a setting for the **hibernate.transaction.coordinator_class** property, Hibernate will automatically build the proper transaction coordinator based on the transaction type for the persistence unit.

If a non-JPA application does not provide a setting for the **hibernate.transaction.coordinator_class** property, Hibernate will default to **jdbc** to manage the transactions. This default will cause problems if the application actually uses JTA-based transactions. A non-JPA application that uses JTA-based transactions should explicitly set the **hibernate.transaction.coordinator_class** property value to **jta** or provide a custom **org.hibernate.resource.transaction.TransactionCoordinatorBuilder** that builds a **org.hibernate.resource.transaction.TransactionCoordinator** that properly coordinates with JTA-based transactions.

**Other Hibernate ORM 5 Changes**

- The **cfg.xml** files are again fully parsed and integrated with events, security, and other functions.
- The properties loaded from the **cfg.xml** using the **EntityManagerFactory** did not previously prefix names with **hibernate**. This has now been made consistent.
- The configuration is no longer serializable.
- The **org.hibernate.dialect.Dialect.getQuerySequencesString()** method now retrieves catalog, schema, and increment values.
- The **AuditConfiguration** modifier was removed from **org.hibernate.envers.boot.internal.EnversService**.
- The **AuditStrategy** method parameters were changed to remove the obsolete **AuditConfiguration** and use the new **EnversService**.
- Various classes and interfaces in the **org.hibernate.hql.spi** package and subpackages have been moved to the new **org.hibernate.hql.spi.id** package. This includes the **MultiTableBulkIdStrategy** class and the **AbstractTableBasedBulkIdHandler**, **TableBasedDeleteHandlerImpl**, and **TableBasedUpdateHandlerImpl** interfaces and their subclasses.
- There was a complete redesign of property access contracts.
- Valid **hibernate.cache.default_cache_concurrency_strategy** setting values are now defined using the **org.hibernate.cache.spi.access.AccessType.getExternalName()** method rather than the **org.hibernate.cache.spi.access.AccessType** enum constants. This is more consistent with other Hibernate settings.

### 5.8.4. Migrating from Hibernate ORM 5.0 to Hibernate ORM 5.1

JBoss EAP 7.1 included Hibernate ORM 5.1. This section highlights the differences and the changes needed when migrating from Hibernate ORM version 5.0 to version 5.1.

**Hibernate ORM 5.1 Features**

This release of Hibernate includes many performance improvements and bug fixes, which are detailed in Hibernate ORM 5.1 Features in the JBoss EAP *7.1.0 Release Notes*. For additional information about the changes implemented between Hibernate ORM 5.0 and Hibernate ORM 5.1, see the Hibernate ORM 5.1 Migration Guide.

**Schema Management Tooling Changes**

**Schema Management Tooling Changes in JBoss EAP 7**

Schema management tooling changes in Hibernate ORM 5.1 are mainly focused in the following areas:

- Unifying the handling of **hbm2ddl.auto** and Hibernate's JPA **schema-generation** support.
- Removing JDBC concerns from the SPI to facilitate true replacement for Hibernate OGM, a persistence engine that provides Java Persistence (JPA) support for NoSQL data stores.

The schema management tooling changes should only be a migration concern for applications that directly use any of the following classes:

- **org.hibernate.tool.hbm2ddl.SchemaExport**
- **org.hibernate.tool.hbm2ddl.SchemaUpdate**
- **org.hibernate.tool.hbm2ddl.SchemaValidator**
- **org.hibernate.tool.schema.spi.SchemaManagementTool**, or any of its delegates

**Schema Management Tooling Changes in JBoss EAP 7.1**

Hibernate ORM 5.1.10 which is included in JBoss EAP 7.1, introduced a new strategy for retrieving database tables that improves **SchemaMigrator** and **SchemaValidator** performance. This strategy executes a single **java.sql.DatabaseMetaData#getTables(String, String, String, String[])** call to determine if each **javax.persistence.Entity** has a mapped database table. This is the default strategy, and it uses the **hibernate.hbm2ddl.jdbc_metadata_extraction_strategy=grouped** property setting. This strategy might require **hibernate.default_schema** and/or **hibernate.default_catalog** to be provided.

To use the old strategy, which executes a **java.sql.DatabaseMetaData#getTables(String, String, String, String[])** call for **each javax.persistence.Entity**, use the **hibernate.hbm2ddl.jdbc_metadata_extraction_strategy=individually** property setting.

### 5.8.5. Migrating from Hibernate ORM 5.1 to Hibernate ORM 5.3

JBoss EAP 7.2 includes Hibernate ORM 5.3. This section highlights some the changes needed when migrating from Hibernate ORM 5.1 to Hibernate ORM 5.3.

**Hibernate ORM 5.2 Features**

Hibernate ORM 5.2 is built using the Java 8 JDK and requires the Java 8 JRE at runtime. The following is a list of some of the changes made in this release.

- The **hibernate-java8** module was merged into **hibernate-core**, and the Java 8 date/time datatypes are now natively supported.
- The **hibernate-entitymanager** module was merged into **hibernate-core**. **HibernateEntityManager** and **HibernateEntityManagerFactory** are deprecated.
- The **Session**, **StatelessSession**, and **SessionFactory** class hierarchies were refactored to remove deprecated classes and to better align with the JPA Metamodel API.

- The SPIs in the **org.hibernate.persister** and **org.hibernate.tuple** packages have changed. Any custom classes using those SPIs will need to be reviewed and updated.
- **LimitHandler** changes introduced a new **hibernate.legacy_limit_handler** setting, which is set to **false** by default, that is designed to allow you to enable the legacy Hibernate 4.3 limit handler behavior. This impacts a limited list of dialects.
- A new strategy for retrieving database tables was introduced that improves **SchemaMigrator** and **SchemaValidator** performance.
- This release changes how **CLOB** values for **String**, **character[]**, and **Character[]** attributes that are annotated with **@Lob** are processed when using PostgreSQL81Dialect and its subclasses.
- The scope of **@TableGenerator** and **@SequenceGenerator** names has changed from global to local.

For the complete list of changes implemented in Hibernate 5.2, see the Hibernate ORM 5.2 Migration Guide.

**Hibernate ORM 5.3 Features**
Hibernate ORM 5.3 adds support for the JPA 2.2 specification. This release contains changes to comply with this specification along with other improvements. The following is a list of some of these changes.

- Changes to positional query parameter handling has resulted in the following changes:
  - Removal of support for JDBC-style parameter declarations in HQL/JPQL queries.
  - JPA positional parameters behave more like named parameters.
  - JDBC-style parameter declarations in native queries use one-based instead of zero-based parameter binding to be consistent with JPA. You can revert back to zero-based binding by setting the **hibernate.query.sql.jdbc_style_params_base** property to **true**.
- To comply with the JPA specification, the sequence value stored by the **@TableGenerator** stored value is that last generated value. Previously, Hibernate stored the next sequence value. You can use the **hibernate.id.generator.stored_last_used** property to enable the legacy Hibernate behavior. Existing applications that use **@TableGenerator** and migrate to Hibernate 5.3 must set the **hibernate.id.generator.stored_last_used configuration** property to **false**.
- The **getType()** method in the **org.hibernate.query.QueryParameter** class was renamed to **getHibernateType()**.
- Hibernate's second-level cache SPI was redesigned to better meet the requirements of the various caching providers. Details can be found in HHH-11356.
- Changes for HHH-11356 also required changes in consumers, which impacts the Hibernate Statistics system.
- Some methods were temporarily added to the **org.hibernate.Query** class to make it easier to migrate native applications from Hibernate ORM 5.1 to 5.3 and maintain the Hibernate 5.1 pagination behavior. These methods are deprecated, and to be portable with future versions of Hibernate, applications should be updated to use the JPA methods.
- Support for using Infinispan as a Hibernate 2nd-level cache provider has been moved to the Infinispan project. As a result, the **hibernate-infinispan** module has been dropped.

- The API of the **org.hibernate.tool.enhance.EnhancementTask** Ant task was changed. The **addFileset()** method was dropped in favor of the **setBase()** and the **setDir()** methods. Details can be found in HHH-11795.

- A bug introduced in Hibernate 4.3 caused many-to-one associations in embeddable collection elements and composite IDs to be eagerly fetched, even when explicitly mapped as lazy. In Hibernate 5.3.2, this bug was fixed. As a result, these associations are fetched as specified by their mappings. Details can be found in HHH-12687.

- JPA and native implementations of Hibernate event listeners were unified in this release. As a result, the **JpaIntegrator** class is obsolete. Classes that extend **org.hibernate.jpa.event.spi.JpaIntegrator** must be modified to have to change these classes to implement the **org.hibernate.integrator.spi.Integrator** interface. Details can be found in HHH-11264.

- The SPIs in the **org.hibernate.persister** package have changed. Any custom classes using those SPIs will need to be reviewed and updated.

For the complete list of these and other changes implemented in Hibernate 5.3, see the Hibernate ORM 5.3 Migration Guide.

#### 5.8.5.1. Exception Handling Changes Between Hibernate 5.1 and Hibernate 5.3

In Hibernate 5.2 and 5.3, exception handling for a **SessionFactory** that is built using Hibernate's native bootstrapping, wraps or converts **HibernateException** according to the JPA specification. The only exception to this behavior is when the operation is Hibernate-specific, for example **Session.save()** or **Session.saveOrUpdate()**.

In Hibernate 5.3.3, the **hibernate.native_exception_handling_51_compliance** property was added. This property indicates whether exception handling for a **SessionFactory** built using Hibernate's native bootstrapping should behave the same as native exception handling in Hibernate ORM 5.1. When set to **true**, **HibernateException** is not wrapped or converted according to the JPA specification. This setting is ignored for a **SessionFactory** built using JPA bootstrapping.

#### 5.8.5.2. Compatibility Transformer

JBoss EAP 7.2 incudes a compatibility transformer that addresses Hibernate ORM 5.3 API methods that are no longer compatible with Hibernate ORM 5.1. The transformer is a temporary measure to allow applications built using Hibernate ORM 5.1 to exhibit the same behavior with Hibernate 5.3 in JBoss EAP 7.2. This is a temporary solution and you should replace these method calls with the recommended JPA method calls.

You can enable the transformer in one of the following ways.

- You can enable the transformer globally for all applications by setting the **Hibernate51CompatibilityTransformer** system property to **true**.

- You can use the **jboss-deployment-structure.xml** file to enable the transformer at the application level.

```
<jboss-deployment-structure>
  <deployment>
    <transformers>
      <transformer class="org.jboss.as.hibernate.Hibernate51CompatibilityTransformer"/>
    </transformers>
  </deployment>
```

```
    <sub-deployment name="main.war">
      <transformers>
        <transformer class="org.jboss.as.hibernate.Hibernate51CompatibilityTransformer"/>
      </transformers>
    </sub-deployment>
  </jboss-deployment-structure>
```

The following table lists the Hibernate 5.1 methods that are transformed and the Hibernate 5.3 method it is converted to.

| Hibernate 5.1 Reference or Method | Transformed to Hibernate 5.3 Reference or Method |
|---|---|
| org.hibernate.BasicQueryContract.getFlushMode() | org.hibernate.BasicQueryContract.getHibernateFlushMode() |
| org.hibernate.Session.getFlushMode() | org.Session.getHibernateFlushMode() |
| Enum org.hibernate.FlushMode.NEVER (0) | Enum org.hibernate.FlushMode.MANUAL (0) |
| org.hibernate.Query.getMaxResults() | org.hibernate.Query.getHibernateMaxResults() |
| org.hibernate.Query.setMaxResults(int) | org.hibernate.Query.setHibernateMaxResults(int) |
| org.hibernate.Query.getFirstResult(int) | org.hibernate.Query.getHibernateFirstResult() |
| org.hibernate.Query.setFirstResult(int) | org.hibernate.Query.setHibernateFirstResult(int) |

## 5.9. HIBERNATE SEARCH CHANGES

The version of Hibernate Search that ships with JBoss EAP 7 has changed. The previous release of JBoss EAP shipped with Hibernate Search 4.6.x. JBoss EAP 7 ships with Hibernate Search 5.5.x.

Hibernate Search 5.5 is built upon Apache Lucene 5.3.1. If you use any native Lucene APIs, be sure to align with this version. The Hibernate Search 5.5 API wraps and hides the complexity of many of the Lucene API changes made between version 3 and version 5; however, some classes are now deprecated, renamed, or repackaged. This section describes how these changes might impact your application code.

### Hibernate Search Mapping Changes

#### Indexing of id Fields of Embedded Relations

When using an **@IndexedEmbedded** annotation to include fields from a related entity, the **id** of the related entity is no longer included. You can enable the inclusion of the **id** by using the **includeEmbeddedObjectId** attribute of the **@IndexedEmbedded** annotation.

#### Example: @IndexedEmbedded Annotation

```
@IndexedEmbedded(includeEmbeddedObjectId=true)
```

#### Number and Date Index Formatting Changes

Numbers and dates are now indexed as numeric fields by default. Properties of type **int**, **long**, **float**, **double**, and their corresponding wrapper classes are no longer indexed as strings. Instead, they are now indexed using Lucene's appropriate numeric encoding. The **id** fields are an exception to this rule. Even when they are represented by a numeric type, they are still indexed as a string keyword by default. The use of **@NumericField** is now obsolete unless you want to specify a custom precision for the numeric encoding. You can keep the old string-based index format by explicitly specifying a string encoding field bridge. In the case of integers, this is the org.hibernate.search.bridge.builtin.IntegerBridge. Check the org.hibernate.search.bridge.builtin package for other publicly available field bridges.

**Date** and **Calendar** are no longer indexed as strings. Instead, instances are encoded as long values representing the number of milliseconds since January 1, 1970, 00:00:00 GMT. You can switch the indexing format by using the new EncodingType enum. For example:

### Example: @DateBridge and @CalendarBridge Annotation

```
@DateBridge(encoding=EncodingType.STRING)
@CalendarBridge(encoding=EncodingType.STRING)
```

The encoding change for numbers and dates is important and can have a big impact on application behavior. If you have a query that targets a field that was previously string-encoded, but is now encoded numerically, you must update the query. Numeric fields must be searched with a **NumericRangeQuery**. You must also make sure that all fields targeted by faceting are string encoded. If you use the Search query DSL, the correct query should be created automatically for you.

### Miscellaneous Hibernate Search Changes

- Sorting options have improved and field encoding specified incorrectly for sorting options now results in runtime exceptions. Lucene also offers more performant sorting if the fields used in the sort are known up front. Hibernate Search 5.5 provides the new **@SortableField** annotation and its multi-valued companion **@SortableFields**. See the Migration Guide from Hibernate Search 5.4 to 5.5 for more information.

- The Lucene **SortField** API requires the following application code change.
  In the previous release of JBoss EAP, you set the type of the sort field in the query as follows.

  ```
  fulltextQuery.setSort(new Sort(new SortField("title", SortField.STRING)));
  ```

  The following is an example of how you set it in JBoss EAP 7.

  ```
  fulltextQuery.setSort(new Sort(new SortField("title", SortField.Type.STRING)));
  ```

- Since **SearchFactory** should only be used by ORM integration, it was moved from the **hibernate-search-engine** module to the **hibernate-search-orm** module. Other integrators should depend exclusively on **SearchIntegrator**, which replaces the deprecated **SearchFactoryIntegrator**.

- The enum value **SpatialMode.GRID** was renamed to **SpatialMode.HASH**.

- **FullTextIndexEventListener** is now a final class. If you currently extend this class, you must find an alternate solution to achieve the same functionality.

- The **hibernate-search-analyzers** module was removed. The recommended approach is to directly use the appropriate Lucene artifact, for example **org.apache.lucene:lucene-analyzers-common**.

- The JMS controller API has changed. The JMS back-end dependency on Hibernate ORM was removed so that it could be used in other non-ORM environments. A consequence is that implementors of **org.hibernate.search.backend.impl.jms.AbstractJMSHibernateSearchController** must adjust to the new signature. This class is an internal class and it is recommended to use it as an example instead of extending it.

- The **org.hibernate.search.spi.ServiceProvider** SPI was refactored. If you were integrating with the old service contract, refer to the Hibernate Search 5.5 Javadoc of **ServiceManager**, **Service**, **Startable** and **Stoppable** for details about the new contract.

- If you have kept indexes generated by Lucene 3.x and have not rebuilt them with Hibernate Search 5.0 or later, you will get an **IndexFormatTooOldException**. It is recommended that you rebuild the indexes with the mass indexer. If you are not able to do that, try to use Lucene's **IndexUpgrader**. You must carefully update the Hibernate Search mappings in case the default behavior has changed. For more information, see the Apache Lucene Migration Guide .

- Apache Lucene was upgraded from 3.6 to 5.3 in JBoss EAP 7. If your code imports Lucene code directly, see the Apache Lucene Migration Guide for details of the changes. Additional information can also be found in the Lucene Change Log.

- When using **@Field(indexNullAs=)** to encode a null marker value in the index, the type of the marker must be compatible with all other values that are indexed in that same field. For example, it was previously possible to encode a null value for numeric fields using a string "null". This is no longer allowed. Instead, you must choose a number to represent the **null** value, such as **-1**.

- Significant improvements were made to the faceting engine. Most of the changes do not affect the API. The one notable exception is that you must now annotate any fields you intend to use for faceting with the **@Facet** or **@Facets** annotation.

### Hibernate Search Renamed and Repackaged Classes

The following is a list of Hibernate Search classes that were repackaged or renamed.

| Previous Package and Class | New Package and Class |
|---|---|
| org.hibernate.search.Environment | org.hibernate.search.cfg.Environment |
| org.hibernate.search.FullTextFilter | org.hibernate.search.filter.FullTextFilter |
| org.hibernate.search.ProjectionConstants | org.hibernate.search.engine.ProjectionConstants |
| org.hibernate.search.SearchException | org.hibernate.search.exception.SearchException |
| org.hibernate.search.Version | org.hibernate.search.engine.Version |

### Lucene - Renamed and Repackaged Classes

Query parsers were moved to a new module, resulting in a packaging change from **org.apache.lucene.queryParser.QueryParser** to **org.apache.lucene.queryparser.classic.QueryParser**.

Many of the Lucene analyzers were refactored, resulting in packaging changes. See the Apache Lucene Documentation to find the replacement packages.

Some Apache Solr utility classes, for example **TokenizerFactory** or **TokenFilterFactory**, were moved into Apache Lucene. If your application uses those utilities or custom analyzers, you must find the new package name in Apache Lucene.

See the Apache Lucene Migration Guide for more information.

### Hibernate Search Deprecated APIs

For the complete list of Hibernate Search deprecated interfaces, classes, enums, annotation types, methods, constructors, and enum constants, see the Hibernate Search Deprecated API document.

#### Hibernate Search Deprecated Interfaces

| Interface | Description |
|---|---|
| org.hibernate.search.store.IndexShardingStrategy | Deprecated as of Hibernate Search 4.4. Might be removed in Search 5. Use **ShardIdentifierProvider** instead. |
| org.hibernate.search.store.Workspace | This interface will be moved and should be considered non-public API. For more information, see HSEARCH-1915. |

#### Hibernate Search Deprecated Classes

| Class | Description |
|---|---|
| org.hibernate.search.filter.FilterKey | Custom filter keys are deprecated and are scheduled for removal in Hibernate Search 6. As of Hibernate Search 5.1, keys for caching Lucene filters are calculated automatically based on the given filter parameters. |
| org.hibernate.search.filter.StandardFilterKey | Custom filter keys are deprecated and are scheduled for removal in Hibernate Search 6. As of Hibernate Search 5.1, keys for caching Lucene filters are calculated automatically based on the given filter parameters. |

#### Hibernate Search Deprecated Enums

| Enum | Description |
|---|---|
| org.hibernate.search.annotations.FieldCacheType | Remove the **CacheFromIndex** annotation as it is deprecated. See Hibernate Search Deprecated Annotations. |

#### Hibernate Search Deprecated Annotations

| Annotation | Description |
|---|---|
| org.hibernate.search.annotations.CacheFromIndex | Remove the annotation. No alternative replacement is necessary. |

| Annotation | Description |
|---|---|
| org.hibernate.search.annotations.Key | Custom filter cache keys are a deprecated feature and are scheduled to be removed in Hibernate Search 6. As of Hibernate Search 5.1, the filter cache keys are determined automatically based on the filter parameters so it is no longer required to provide a key object. |

### Hibernate Search Deprecated Methods

| Method | Description |
|---|---|
| org.hibernate.search.FullTextSharedSessionBuilder.autoClose() | No replacement |
| org.hibernate.search.FullTextSharedSessionBuilder.autoClose(boolean) | No replacement |
| org.hibernate.search.cfg.IndexedMapping.cacheFromIndex(FieldCacheType…) | This will be removed with no replacement. |
| org.hibernate.search.cfg.EntityDescriptor.getCacheInMemory() | This will be removed with no replacement. |
| org.hibernate.search.cfg.ContainedInMapping.numericField() | Invoke **field().numericField()** instead. |
| org.hibernate.search.cfg.EntityDescriptor.setCacheInMemory(Map<String, Object>) | This will be removed with no replacement. |
| org.hibernate.search.MassIndexer.threadsForSubsequentFetching(int) | This method will be removed. |
| org.hibernate.search.query.dsl.FuzzyContext.withThreshold(float) | Use **FuzzyContext.withEditDistanceUpTo(int)**. |

### Hibernate Search Deprecated Constructors

| Constructor | Description |
|---|---|
| org.hibernate.search.cfg.NumericFieldMapping(PropertyDescriptor, EntityDescriptor, SearchMapping) | Use **NumericFieldMapping.NumericFieldMapping(String, PropertyDescriptor, EntityDescriptor, SearchMapping)** instead. |

### Changes Impacting Advanced Integrators

This section describes changes that are not part of the public API. They should not impact the average developer as these artifacts should only be accessed by integrators who extend the Hibernate Search framework.

- The **IndexWriterSetting.MAX_THREAD_STATES** and **IndexWriterSetting.TERM_INDEX_INTERVAL** enum constants are deprecated. They affect which properties are read from the configuration, so the fact they they are missing means that configuration properties such as **hibernate.search.Animals.2.indexwriter.term_index_interval = default** are now ignored. The only side effect is that the property is not applied.

- The **SearchFactoryIntegrator** interface is now deprecated. You should immediately migrate all code to use **SearchIntegrator**.

- The **SearchFactoryBuilder** class is now deprecated. Use **SearchIntegrationBuilder** instead.

- The **HSQuery.getExtendedSearchIntegrator()** method has been deprecated. It might be possible to use **SearchIntegrator**, but it is preferable to remove it altogether.

- The **DocumentBuilderIndexedEntity.getFieldCacheOption()** method has been deprecated. There is no replacement.

- The **BuildContext.getIndexingStrategy()** method is deprecated. Use **BuildContext.getIndexingMode()** instead.

- The **DirectoryHelper.getVerifiedIndexDir(String, Properties, boolean)** method is deprecated. Use **DirectoryHelper.getVerifiedIndexPath(java.lang.String, java.util.Properties, boolean)** instead.

- The following is a list of Hibernate Search classes that were repackaged or renamed.

| Previous Package and Class | New Package and Class |
|---|---|
| org.hibernate.search.engine.impl.SearchMappingBuilder | org.hibernate.search.engine.spi.SearchMappingHelper |
| org.hibernate.search.indexes.impl.DirectoryBasedIndexManager | org.hibernate.search.indexes.spi.DirectoryBasedIndexManager |
| org.hibernate.search.spi.MassIndexerFactory | org.hibernate.search.batchindexing.spi.MassIndexerFactory |
| org.hibernate.search.spi.SearchFactoryBuilder | org.hibernate.search.spi.SearchIntegratorBuilder |
| org.hibernate.search.spi.SearchFactoryIntegrator | org.hibernate.search.spi.SearchIntegrator |

## 5.10. MIGRATE ENTITY BEANS TO JPA

Support for EJB Entity Beans is optional in Java EE 7 and they are not supported in JBoss EAP 7. This means container-managed persistence (CMP) and bean-managed persistence (BMP) entity beans must be rewritten to use Java Persistence API (JPA) entities when you migrate to JBoss EAP 7.

In previous releases of JBoss EAP, entity beans were created in application source code by extending the **javax.ejb.EntityBean** class and implementing the required methods. They were then configured in the **ejb-jar.xml** file. A CMP entity bean was specified using an **<entity>** element that contained a **<persistence-type>** child element with a value of **Container**. A BMP entity bean was specified using an **<entity>** element that contained a **<persistence-type>** child element with a value of **Bean**.

In JBoss EAP 7, you must replace any CMP and BMP entity beans in your code with Java Persistence API (JPA) entities. JPA entities are created using the javax.persistence.* classes and are defined in the **persistence.xml** file.

The following is an example of a JPA entity class.

```
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
// User is a keyword in some SQL dialects!
@Table(name = "MyUsers")
public class MyUser {
    @Id
    @GeneratedValue
    private Long id;

    @Column(unique = true)
    private String username;
    private String firstName;
    private String lastName;

    public Long getId() {
        return id;
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getFirstName() {
        return firstName;
    }
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
    public String getLastName() {
        return lastName;
    }
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
```

The following is an example of a **persistence.xml** file.

```
<persistence version="2.1"
      xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
      xsi:schemaLocation="
        http://xmlns.jcp.org/xml/ns/persistence
        http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
```

```
  <persistence-unit name="my-unique-persistence-unit-name">
    <properties>
      // properties...
    </properties>
  </persistence-unit>
</persistence>
```

For working examples of JPA entities, see the **bmt**, **cmt**, and **hibernate5** quickstarts that ship with JBoss EAP 7.

## 5.11. JPA PERSISTENCE PROPERTY CHANGES

### JPA Persistence Property Changes Introduced in JBoss EAP 7.0

A new persistence property, **jboss.as.jpa.deferdetach**, was added to provide compatibility with the persistence behavior in previous releases of JBoss EAP.

The **jboss.as.jpa.deferdetach** property controls whether the transaction-scoped persistence context used in a non-JTA transaction thread detaches loaded entities after each **EntityManager** invocation or whether it waits until the persistence context is closed, for example, when the session bean invocation ends. The property value defaults to **false**, meaning entities are detached or cleared after each **EntityManager** invocation. This is the correct default behavior as defined in the JPA specification. If the property value is set to **true**, the entities are not detached until the persistence context is closed.

In JBoss EAP 5, persistence behaved as if the **jboss.as.jpa.deferdetach** property was set to **true**. To get this same behavior when migrating your application from JBoss EAP 5 to JBoss EAP 7, you must set the **jboss.as.jpa.deferdetach** property value to **true** in your **persistence.xml** as shown in the following example.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence" version="1.0">
    <persistence-unit name="EAP5_COMPAT_PU">
    <jta-data-source>java:jboss/datasources/ExampleDS</jta-data-source>
    <properties>
         <property name="jboss.as.jpa.deferdetach" value="true" />
    </properties>
  </persistence-unit>
</persistence>
```

In JBoss EAP 6, persistence behaved as if the **jboss.as.jpa.deferdetach** property was set to **false**. This is the same behavior as seen in JBoss EAP 7, so no changes are necessary when you migrate your application.

### JPA Persistence Property Changes Introduced in JBoss EAP 7.1

In JBoss EAP 7.0, unsynchronized persistence context error checking was not as strict as it should have been in the following areas.

- A synchronized container-managed persistence context was allowed to use an unsynchronized extended persistence context that was associated with a JTA transaction. Instead, it should have thrown an **IllegalStateException** to prevent the unsynchronized persistence context from being used.
- An unsynchronized persistence context specified in a deployment descriptor was treated as synchronized.

In addition, **PersistenceProperty** hints in the **@PersistenceContext** were mistakenly ignored in JBoss EAP 7.0.

These issues were addressed and fixed in JBoss EAP 7.1 and later. Because these updates can result in an unwanted change in application behavior, two new persistence unit properties were introduced in JBoss EAP 7.1 to provide backward compatibility and preserve the previous behavior.

| Property | Description |
|---|---|
| **wildfly.jpa.skipmixedsynctypechecking** | This property disables the error checking. It should only be used as a temporary measure for backward compatibility in situations where applications worked in JBoss EAP 7.0 and fail in JBoss EAP 7.1 and later. Because this property might be deprecated in a future release, it is recommended that you correct your application code as soon as you are able to do so. |
| **wildfly.jpa.allowjoinedunsync** | This property is an alternative to **wildfly.jpa.skipmixedsynctypechecking**. It allows the application to treat unsynchronized persistence contexts that are associated with a JTA transaction as if they are synchronized persistence contexts. |

## 5.12. MIGRATE EJB CLIENT CODE

### 5.12.1. EJB Client Changes in JBoss EAP 7

The default remote connector and port has changed in JBoss EAP 7. For details about this change, see Update the Remote URL Connector and Port .

If you used the **migrate** operation to migrate your server configuration, the old settings are preserved and you do not need to make the changes detailed below. However, if you run with the new JBoss EAP 7 default configuration, you must make the following changes.

#### 5.12.1.1. Update the Default Remote Connection Port

Change the remote connection port value from **4447** to **8080** in the **jboss-ejb-client.properties** file.

The following are examples of a **jboss-ejb-client.properties** file in the previous and the current release.

**Example: JBoss EAP 6 jboss-ejb-client.properties File**

```
remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABLED=false
remote.connections=default
remote.connection.default.host=localhost
remote.connection.default.port=4447
remote.connection.default.connect.options.org.xnio.Options.SASL_POLICY_NOANONYMOUS=false
```

**Example: JBoss EAP 7 jboss-ejb-client.properties File**

```
remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABLED=false
remote.connections=default
remote.connection.default.host=localhost
remote.connection.default.port=8080
remote.connection.default.connect.options.org.xnio.Options.SASL_POLICY_NOANONYMOUS=false
```

#### 5.12.1.2. Update the Default Connector

If you are running with the new JBoss EAP 7 configuration, the default connector has changed from **remote** to **http-remoting**. This change impacts clients that use libraries from one release of JBoss EAP and to connect to server in a different release.

- If a client application uses the EJB client library from JBoss EAP 6 and wants to connect to JBoss EAP 7 server, the server must be configured to expose a **remote** connector on a port other than **8080**. The client must then connect using that newly configured connector.
- A client application that uses the EJB client library from JBoss EAP 7 and wants to connect to JBoss EAP 6 server must be aware that the server instance does not use the **http-remoting** connector and instead uses a **remote** connector. This is achieved by defining a new client-side connection property.

  **Example: remote Connection Property**

  ```
  remote.connection.default.protocol=remote
  ```

### 5.12.2. Migrate Remote Naming Client Code

If you are running with the new default JBoss EAP 7 configuration, you must modify your client code to use the new default remote port and connector.

The following is an example of how remote naming properties were specified in the client code in JBoss EAP 6.

```
java.naming.factory.initial=org.jboss.naming.remote.client.InitialContextFactory
java.naming.provider.url=remote://localhost:4447
```

The following is an example of how to specify the remote naming properties in the client code in JBoss EAP 7.

```
java.naming.factory.initial=org.wildfly.naming.client.WildFlyInitialContextFactory
java.naming.provider.url=http-remoting://localhost:8080
```

### 5.12.3. Additional EJB Client Changes Introduced in JBoss EAP 7.1

While JBoss EAP 7.0 shipped with JBoss EJB Client 2.1.4, JBoss EAP 7.1 and later ship with JBoss EJB Client 4.0.x, which includes a number of changes to the API.

- The **org.ejb.client.EJBClientInvocationContext** class has added the following new methods.

| Method | Type | Description |
|---|---|---|

| Method | Type | Description |
|---|---|---|
| **isBlockingCaller()** | boolean | Determine whether this invocation is currently blocking the calling thread. |
| **isClientAsync()** | boolean | Determine whether the method is marked client-asynchronous, meaning that invocation should be asynchronous regardless of whether the server-side method is asynchronous. |
| **isIdempotent()** | boolean | Determine whether the method is marked idempotent, meaning that the method may be invoked more than one time with no additional effect. |
| **setBlockingCaller(boolean)** | void | Establish whether this invocation is currently blocking the calling thread. |
| **setLocator(EJBLocator<T>)** | **<T> void** | Set the locator for the invocation target. |

- The **org.ejb.client.EJBLocator** class has added the following new methods.

| Method | Type | Description |
|---|---|---|
| **asStateful()** | **StatefulEJBLocator<T>** | Return this locator as a stateful locator, if it is one. |
| **asStateless()** | **StatelessEJBLocator<T>** | Return this locator as a stateless locator, if it is one. |
| **isEntity()** | boolean | Determine if this is an entity locator. |
| **isHome()** | boolean | Determine if this is a home locator. |
| **isStateful()** | boolean | Determine if this is a stateful locator. |
| **isStateless()** | boolean | Determine if this is a stateless locator. |
| **withNewAffinity(Affinity)** | **abstract EJBLocator<T>** | Create a copy of this locator, but with the new given affinity. |

- A new **org.ejb.client.EJBClientPermission** class, which is a subclass of **java.security.Permission**, has been introduced for controlling access to privileged EJB operations.
  - It provides the following constructors.
    - **EJBClientPermission(String name)**

- **EJBClientPermission(String name, String actions)**

- It provides the following methods.

| Method | Type | Description |
|---|---|---|
| **equals(EJBClientPermission obj)** | boolean | Checks two **EJBClientPermission** objects for equality. |
| **equals(Object obj)** | boolean | Checks two **Permission** objects for equality. |
| **equals(Permission obj)** | boolean | Checks two **Permission** objects for equality. |
| **getActions()** | String | Returns the actions as a string. |
| **hashcode()** | int | Returns the hash code value for this **Permission** object. |
| **implies(EJBClientPermission permission)** | boolean | Checks if the specified permission's actions are *implied by* this **EJBClientPermission** object's actions. |
| **implies(Permission permission)** | boolean | Checks if the specified permission's actions are *implied by* this **Permission** object's actions. |

- A new **org.ejb.client.EJBMethodLocator** class has been introduced for locating a specific EJB method.
  - It provides the following constructor.
    - **EJBMethodLocator(String methodName, String… parameterTypeNames)**
  - It provides the following methods.

| Method | Type | Description |
|---|---|---|
| **equals(EJBMethodLocator other)** | boolean | Determine whether this object is equal to another. |
| **equals(Object other)** | boolean | Determine whether this object is equal to another. |
| **forMethod(Method method)** | **static EJBMethodLocator** | Get a method locator for the given reflection method. |
| **getMethodName()** | String | Get the method name. |
| **getParameterCount()** | int | Get the parameter count. |

| Method | Type | Description |
|---|---|---|
| **getParameterTypeName(int index)** | String | Get the name of the parameter at the given index. |
| **hashCode()** | int | Get the hash code. |

- A new **org.jboss.ejb.client.EJBReceiverInvocationContext.ResultProducer.Failed** class has been introduced for failure cases.
    - It provides the following constructor.
        - **Failed(Exception cause)**
    - It provides the following methods.

| Method | Type | Description |
|---|---|---|
| **discardResult()** | void | Discard the result, indicating that it will not be used. |
| **getResult()** | **Object** | Get the result. |

- A new **org.jboss.ejb.client.EJBReceiverInvocationContext.ResultProducer.Immediate** class has been introduced for immediate results.
    - It provides the following constructor.
        - **Failed(Exception cause)**
    - It provides the following methods.

| Method | Type | Description |
|---|---|---|
| **discardResult()** | void | Discard the result, indicating that it will not be used. |
| **getResult()** | **Object** | Get the result. |

- A new **org.jboss.ejb.client.URIAffinity** class, which is a subclass of **org.jboss.ejb.client.Affinity** has been introduced for URI affinity specification.
    - It is created using **Affinity.forUri(URI)**.
    - It provides the following methods.

| Method | Type | Description |
|---|---|---|
| **equals(Affinity other)** | boolean | Indicates whether another object is equal to this one. |

| Method | Type | Description |
|---|---|---|
| **equals(Object other)** | boolean | Indicates whether another object is equal to this one. |
| **equals(URIAffinity other)** | boolean | Indicates whether another object is equal to this one. |
| **getURI()** | URI | Get the associated URI. |
| **hashCode()** | int | Get the hash code. |
| **toString()** | String | Returns a string representation of the object. |

- The **org.jboss.ejb.client.EJBMetaDataImpl** class has deprecated the following methods.
  - **toAbstractEJBMetaData()**
  - **EJBMetaDataImpl(AbstractEJBMetaData<?,?>)**

### 5.12.4. EJB Client Changes Needed for JBoss EAP 7.2

The upgrade of the **org.apache.santuario.xmlsec** module from 2.0.8 to 2.1.1 in JBoss EAP 7.2 has caused a regression with remoting in **PicketLinkSTS**. The issue manifests itself as the following runtime exception.

```
java.lang.IllegalArgumentException: ELY05131: Invalid ASCII control "0xA"
```

This is caused by a change in the formatting in the generated **SignatureValue** in the assertion. In the previous release, the generated value looked similar to the following example.

```
<dsig:SignatureValue
xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">cUNpFJIZlLYrBDZtQSTDrq2K6PbnAHyg2qbx/D5F
uB4XMjdQ5oxQjkMejLyelnA7s4GFusoLhahlqlTOT8UrOyxrR4yYAmJ/e5s+f4gys926+tbiraT/3/wG8wM/L
vcjvk5Ap69zODuRYpypsWfA4jrI7TTBXVPGy8g4KUdnFviUiTuFTc2Ghgxp53AmUuLis/THyP28jE7+28//
q8bi/bQrFwHC6tWX67+NK1duFCOcQ6IPIKeVrePZz55Ivgl+WWdkF6uYCz5IdMzurhzmeQ3K8DAMIxz/
MG67VWJIOnuGNWF7nmdye5zd9AFcRsr1XadvZJCbGNfuc89AL5inCg==</dsig:SignatureValue>
```

In JBoss EAP 7.2, the generated string value now contains instances of invalid hidden ASCII **0xD** carriage return and **0xA** line feed control characters.

```
<dsig:SignatureValue
xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">cUNpFJIZlLYrBDZtQSTDrq2K6PbnAHyg2qbx/D5F
uB4XMjdQ5oxQjkMejLyelnA7s4GFusoLhahl&#13;
qlTOT8UrOyxrR4yYAmJ/e5s+f4gys926+tbiraT/3/wG8wM/Lvcjvk5Ap69zODuRYpypsWfA4jrI&#13;
7TTBXVPGy8g4KUdnFviUiTuFTc2Ghgxp53AmUuLis/THyP28jE7+28//q8bi/bQrFwHC6tWX67+N&#13;

K1duFCOcQ6IPIKeVrePZz55Ivgl+WWdkF6uYCz5IdMzurhzmeQ3K8DAMIxz/MG67VWJIOnuGNWF7&
#13;
nmdye5zd9AFcRsr1XadvZJCbGNfuc89AL5inCg==</dsig:SignatureValue>
```

If you experience the above runtime exception, you must update your client code to remove occurrences of the hidden ASCII characters from the the returned assertion string.

For example, assume your current code looks similar to the following example.

```
WSTrustClient client = new WSTrustClient("PicketLinkSTS", "PicketLinkSTSPort",
            "http://localhost:8080/picketlink-sts/PicketLinkSTS", new
WSTrustClient.SecurityInfo(username, password));
Element assertion = client.issueToken(SAMLUtil.SAML2_TOKEN_TYPE);

// Return the assertion as a string
String assertionString = DocumentUtil.getNodeAsString(assertion);
...
properties.put("remote.connection.main.password", assertionString);
```

You must add a line of code to remove the occurrences of the invalid hidden ASCII **0xD** carriage return and **0xA** line feed characters as shown below.

```
WSTrustClient client = new WSTrustClient("PicketLinkSTS", "PicketLinkSTSPort",
            "http://localhost:8080/picketlink-sts/PicketLinkSTS", new
WSTrustClient.SecurityInfo(username, password));
Element assertion = client.issueToken(SAMLUtil.SAML2_TOKEN_TYPE);

// Return the assertion as a string, stripping the invalid hidden ASCII characters
String assertionString = DocumentUtil.getNodeAsString(assertion).replace(String.valueOf((char)
0xA), "").replace(String.valueOf((char) 0xD), "");
...
properties.put("remote.connection.main.password", assertionString);
```

## 5.13. MIGRATE CLIENTS TO USE THE WILDFLY CONFIGURATION FILE

Prior to release 7.1, JBoss EAP client libraries, such as EJB and naming, used different configuration strategies. JBoss EAP 7.1 introduced the **wildfly-config.xml** file with the purpose of unifying all client configurations into one single configuration file, in a similar manner to the way the server configuration is handled.

For example, prior to JBoss EAP 7.1, you might create a new **InitialContext** for a Java EJB client using a **jboss-ejb-client.properties** file, or by programmatically setting the properties using a **Properties** class.

**Example: jboss-ejb-client.properties Properties File**

```
remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABLED=false
remote.connections=one
remote.connection.one.port=8080
remote.connection.one.host=127.0.0.1
remote.connection.one.username=quickuser
remote.connection.one.password=quick-123
```

In JBoss EAP 7.1 and later, you create a **wildfly-config.xml** file in the **META-INF/** directory of the client archive. This is the equivalent configuration using a **wildfly-config.xml** file.

**Example: Equivalent Configuration Using the wildfly-config.xml File**

```
<configuration>
```

```
    <authentication-client xmlns="urn:elytron:client:1.2">
        <authentication-rules>
            <rule use-configuration="ejb"/>
        </authentication-rules>
        <authentication-configurations>
            <configuration name="ejb">
                <set-user-name name="quickuser"/>
                <credentials>
                    <clear-password password="quick-123"/>
                </credentials>
            </configuration>
        </authentication-configurations>
    </authentication-client>
    <jboss-ejb-client xmlns="urn:jboss:wildfly-client-ejb:3.0">
        <connections>
            <connection uri="remote+http://127.0.0.1:8080" />
        </connections>
    </jboss-ejb-client>
</configuration>
```

For information about how to configure client authentication for the Elytron Client using the **wildfly-config.xml** file, see Configure Client Authentication with Elytron Client in *How to Configure Identity Management* for JBoss EAP.

For more information about the types of client configurations that can be done using the **wildfly-config.xml** file, see Client Configuration Using the **wildfly-config.xml** File in the JBoss EAP *Development Guide*.

## 5.14. MIGRATE DEPLOYMENT PLAN CONFIGURATIONS

The Java EE Application Deployment specification (JSR-88) was intended to define a standard contract to enable tools from multiple providers to configure and deploy applications on any Java EE platform product. The contract required Java EE Product Providers to implement the **DeploymentManager** and other **javax.enterprise.deploy.spi** interfaces to be accessed by the Tool Providers. In case of JBoss EAP 6, a deployment plan is identified by an XML descriptor named **deployment-plan.xml** that is bundled in a ZIP or JAR archive.

This specification saw very little adoption because most application server products provide their own more "feature rich" deployment solutions. For this reason, JSR-88 support was dropped from Java EE 7 and, in turn, from JBoss EAP 7.

If you used JSR-88 to deploy your application, you must now use another method to deploy the application. The JBoss EAP management CLI **deploy** command provides a standard way to deploy archives to standalone servers or to server groups in a managed domain. For more information about the management CLI, see the Management CLI Guide.

## 5.15. MIGRATE CUSTOM APPLICATION VALVES

You must manually migrate custom valves or any valves that are defined in the **jboss-web.xml** XML file. This includes valves created by extending the **org.apache.catalina.valves.ValveBase** class and configured in the **<valve>** element of the **jboss-web.xml** descriptor file.

IMPORTANT

Custom valves and valves that are defined in the **jboss-web.xml** file must be rewritten or replaced by the corresponding Undertow built-in handler. For information about mapping valves to Undertow handlers, see Migrate JBoss Web Valves.

Authentication valves must be replaced manually using Undertow built-in authentication mechanisms.

#### Migrate Valves Configured in Deployments

In JBoss EAP 6, you could define custom valves at the application level by configuring them in the **jboss-web.xml** web application descriptor file. In JBoss EAP 7, it is possible to do this with Undertow handlers as well.

The following is an example of a valve configured in the **jboss-web.xml** file in JBoss EAP 6.

```
<jboss-web>
    <valve>
        <class-name>org.jboss.examples.MyValve</class-name>
        <param>
            <param-name>myParam</param-name>
            <param-value>foobar</param-value>
        </param>
    </valve>
</jboss-web>
```

For more information about how to create and configure custom handlers in JBoss EAP, see Creating Custom Handlers in the JBoss EAP *Development Guide*.

#### Migrate Custom Authenticator Valves

For information about how to migrate authenticator valves, see Migrate Authenticator Valves in this guide.

## 5.16. SECURITY APPLICATION CHANGES

The replacement of JBoss Web with Undertow requires changes to security configuration in JBoss EAP 7.

### 5.16.1. Migrate Authenticator Valves

If you created a custom authenticator valve that extended **AuthenticatorBase** in JBoss EAP 6.4, you must manually replace it with a custom HTTP authentication implementation in JBoss EAP 7. The HTTP authentication mechanism is created in the **elytron** subsystem and then registered with the **undertow** subsystem. For information about how to implement a custom HTTP authentication mechanism, see Developing a Custom HTTP Mechanism in the *Development Guide* for JBoss EAP.

### 5.16.2. PicketLink Changes

For information about the changes required for SSO with SAML v2 configuration, see Changes from Previous Versions of JBoss EAP in *How To Set Up SSO with SAML v2* for JBoss EAP.

### 5.16.3. Other Security Application Changes

For information about the differences in SSO configuration with Kerberos, see Differences from Configuring Previous Versions JBoss EAP in *How to Set Up SSO with Kerberos* for JBoss EAP.

## 5.17. JBOSS LOGGING CHANGES

If your application uses JBoss Logging, be aware that the annotations in the **org.jboss.logging** package are now deprecated in JBoss EAP 7. They have been moved to the **org.jboss.logging.annotations** package, so you must update your source code to import the new package.

The annotations have also moved to a separate Maven **groupId:artifactId:version** (GAV) ID so you need to add a new project dependency for **org.jboss.logging:jboss-logging-annotations** in your project **pom.xml** file.

### NOTE

Only the logging annotations have moved. The **org.jboss.logging.BasicLogger** and **org.jboss.logging.Logger** still exist in the **org.jboss.logging** package.

The following table lists the deprecated annotation classes and corresponding replacements.

**Table 5.1. Deprecated Logging Annotation Replacements**

| Deprecated Class | Replacement Class |
|---|---|
| org.jboss.logging.Cause | org.jboss.logging.annotations.Cause |
| org.jboss.logging.Field | org.jboss.logging.annotations.Field |
| org.jboss.logging.FormatWith | org.jboss.logging.annotations.FormatWith |
| org.jboss.logging.LoggingClass | org.jboss.logging.annotations.LoggingClass |
| org.jboss.logging.LogMessage | org.jboss.logging.annotations.LogMessage |
| org.jboss.logging.Message | org.jboss.logging.annotations.Message |
| org.jboss.logging.MessageBundle | org.jboss.logging.annotations.MessageBundle |
| org.jboss.logging.MessageLogger | org.jboss.logging.annotations.MessageLogger |
| org.jboss.logging.Param | org.jboss.logging.annotations.Param |
| org.jboss.logging.Property | org.jboss.logging.annotations.Property |

## 5.18. JAVASERVER FACES (JSF) CODE CHANGES

### Dropped Support for JSF 1.2

JBoss EAP 6.4 allowed you to continue to use JSF 1.2 with your application deployment by creating a **jboss-deployment-structure.xml** file.

JBoss EAP 7.2 includes JSF 2.3 and no longer supports the JSF 1.2 API. If your application uses JSF 1.2, you must rewrite it to use JSF 2.3.

## 5.19. MODULE CLASS LOADING CHANGES

In JBoss EAP 7, the class loading behavior has changed in cases where multiple modules contain the same classes or packages.

Assume there are two modules, **MODULE_A** and **MODULE_B**, that depend upon each other and contain some of the same packages. In JBoss EAP 6, the classes or packages that were loaded from the dependencies took precedence over those specified in the **resource-root** of the **module.xml** file. This meant **MODULE_A** saw the packages for **MODULE_B** and **MODULE_B** saw the packages for **MODULE_A**. This behavior was confusing and could cause conflicts. This behavior has changed in JBoss EAP 7. Now the classes or packages specified by the **resource-root** in the **module.xml** file take precedence over those specified by the dependency. This means **MODULE_A** sees the packages for **MODULE_A** and **MODULE_B** sees the packages for **MODULE_B**. This prevents conflicts and provides a more appropriate behavior.

If you have defined custom modules that include **resource-root** libraries or packages that contain classes that are duplicated in their module dependencies, you might see **ClassCastException**, **LinkageError**, class loading errors, or other changes in behavior when you migrate to JBoss EAP 7. To resolve these issues, you must configure your **module.xml** file to ensure only one version of a class is used. This can be accomplished by using either of the following approaches.

- You can avoid specifying a **resource-root** that duplicates classes in the module dependency.
- You can use the **include** and **exclude** sub-elements of the **imports** and **exports** elements to control class loading in the **module.xml** file. The following is an export element that excludes classes is in the specified package.

```
<exports>
  <exclude path="com/mycompany/duplicateclassespath/"/>
</exports>
```

If you prefer to preserve your existing behavior, you must filter the dependency packages from the dependent **resource-root** in the **module.xml** file using the **filter** element. This allows you to retain the existing behavior without the odd looping that you would see under JBoss EAP 6. The following is an example of a **root-resource** that filters classes in a specified package.

```
<resource-root path="mycompany.jar">
  <filter>
    <exclude path="com/mycompany/duplicateclassespath"/>
  </filter>
</resource-root>
```

For more information about modules and class loading, see Class Loading and Modules in the JBoss EAP *Development Guide*.

## 5.20. APPLICATION CLUSTERING CHANGES

### 5.20.1. Overview of New Clustering Features

The following list describes some of the new clustering features to be aware of when migrating your application from JBoss EAP 6 to JBoss EAP 7.

- JBoss EAP 7 introduces a new public API for building singleton services that significantly simplifies the process. For information on singleton services, see HA Singleton Service in the JBoss EAP *Development Guide*
- A singleton deployment can be configured to deploy and start on only a single node in the cluster at a time. For more information, see HA Singleton Deployments in the JBoss EAP *Development Guide*.
- You can now define clustered singleton MDBs. For more information, see Clustered Singleton MDBs in *Developing EJB Applications* for JBoss EAP.
- JBoss EAP 7 includes the Undertow mod_cluster implementation. This offers a pure Java load balancing solution that does not require an httpd web server. For more information, see Configuring JBoss EAP as a Front-end Load Balancer in the JBoss EAP *Configuration Guide*.

The remainder of this section describes how clustering changes might impact the migration of your applications to JBoss EAP 7.

### 5.20.2. Web Session Clustering Changes

JBoss EAP 7 introduces a new web session clustering implementation. It replaces the previous implementation, which was tightly coupled to the legacy JBoss Web subsystem source code.

The new web session clustering implementation impacts how the application is configured in the **jboss-web.xml** JBoss EAP proprietary web application XML descriptor file. The following are the only clustering configuration elements that remain in this file.

```
<jboss-web>
  ...
  <max-active-sessions>...</max-active-sessions>
  ...
  <replication-config>
    <replication-granularity>...</replication-granularity>
    <cache-name>...</cache-name>
  </replication-config>
  ...
</jboss-web>
```

The following table describes how to achieve similar behavior for elements in the **jboss-web.xml** file that are now obsolete.

| Configuration Element | Description of Change |
|---|---|
| <max-active-sessions/> | Previously, the session creation would fail if it caused the number of active sessions to exceed the value specified by **<max-active-sessions/>**.<br><br>In the new implementation, **<max-active-sessions/>** is used to enable session passivation. If session creation will cause the number of active sessions to exceed the **<max-active-sessions/>**, then the oldest session known to the session manager will passivate to make room for the new session. |
| <passivation-config/> | This configuration element and its sub-elements are no longer used in JBoss EAP 7. |

| Configuration Element | Description of Change |
|---|---|
| <use-session-passivation/> | Previously, passivation was enabled using this attribute.<br><br>In the new implementation, passivation is enabled by specifying a non-negative value for **<max-active-sessions/>**. |
| <passivation-min-idle-time/> | Previously, sessions needed to be active for a minimum amount of time before becoming a candidate for passivation. This could cause session creation to fail, even when passivation was enabled.<br><br>The new implementation does not support this logic and thus avoids this Denial of Service (DoS) vulnerability. |
| <passivation-max-idle-time/> | Previously, a session would be passivated after it was idle for a specific amount of time.<br><br>The new implementation only supports lazy passivation. It does not support eager passivation. Sessions are only passivated when necessary to comply with **<max-active-sessions/>**. |
| <replication-config/> | The new implementation deprecates a number of sub-elements. |
| <replication-trigger/> | Previously, this element was used to determine when session replication was triggered. The new implementation replaces this configuration option with a single, robust strategy. For more information, see Immutable Session Attributes in the JBoss EAP *Development Guide*. |
| <use-jk/> | Previously, the **instance-id** of the node handling a given request was appended to the **jsessionid**, for use by load balancers such as mod_jk, mod_proxy_balancer, mod_cluster, depending on the value specified for **<use-jk/>**.<br><br>In the new implementation, the **instance-id**, if defined, is always appended to the **jsessionid**. |
| <max-unreplicated-interval/> | Previously, this configuration option was intended as an optimization to prevent the replication of a session's timestamp if no session attribute was changed. While this sounds nice, in practice it does not prevent any RPCs, since session access requires cache transaction RPCs regardless of whether any session attributes changed.<br><br>In the new implementation, the timestamp of a session is replicated on every request. This prevents stale session metadata following a failover. |
| <snapshot-mode/> | Previously, one could configure **<snapshot-mode/>** as **INSTANT** or **INTERVAL**. Infinispan's asynchronous replication makes this configuration option obsolete. |
| <snapshot-interval/> | This was only relevant for **<snapshot-mode>INTERVAL</snapshot-mode>**. Since **<snapshot-mode/>** is obsolete, this option is now obsolete as well. |

| Configuration Element | Description of Change |
|---|---|
| <session-notification-policy/> | Previously, the value specified by this attribute defined a policy for triggering session events.<br><br>In the new implementation, this behavior is specification-driven and not configurable. |

This new implementation also supports write-through cache stores as well as passivation-only cache stores. Typically, a write-through cache store is used in conjunction with an invalidation cache. The web session clustering implementation in JBoss EAP 6 did not operate correctly when used with an invalidation cache.

### 5.20.3. Stateful Session EJB Clustering Changes

In JBoss EAP 6, you were required to enabled the clustering behavior for stateful session beans (SFSBs) in one of the following ways.

- You could add the **org.jboss.ejb3.annotation.Clustered** annotation in the session bean.

```
@Stateful
@Clustered
public class MyBean implements MySessionInt {

  public void myMethod() {
    //
  }
}
```

- You could add the **<clustered>** element to the **jboss-ejb3.xml** file.

```
<c:clustering>
  <ejb-name>DDBasedClusteredSFSB</ejb-name>
  <c:clustered>true</c:clustered>
</c:clustering>
```

JBoss EAP 7 no longer requires you to enable the clustering behavior. By default, if the server is started using an HA profile, the state of SFSBs will be replicated automatically.

You can disable this default behavior in one of the following ways.

- You can disable the default behavior for a single stateful session bean by using **@Stateful(passivationCapable=false)**, which is new to the EJB 3.2 specification.
- You can disable this behavior globally in the configuration of the **ejb3** subsystem in the server configuration.

NOTE

If the **@Clustered** annotation is not removed from the application, it is simply ignored and does not affect the deployment of the application.

### 5.20.4. Clustering Services Changes

In JBoss EAP 6, the APIs for clustering services were in private modules and were not supported.

JBoss EAP 7 introduces a public clustering services API for use by applications. The new services are designed to be lightweight, easily injectable, and require no external dependencies.

- The new **org.wildfly.clustering.group.Group** interface provides access to the current cluster status and allows listening for cluster membership changes.
- The new **org.wildfly.clustering.dispatcher.CommandDispatcher** interface allows running code in the cluster, on all or a selected subset of nodes.

These services replace similar APIs that were available in previous releases, namely **HAPartition** from JBoss EAP 5 and **GroupCommunicationService**, **GroupMembershipNotifier**, and **GroupRpcDispatcher** in JBoss EAP 6.

For more information, see Public API for Clustering Services in the JBoss EAP *Development Guide*.

### 5.20.5. Migrate Clustering HA Singleton

In JBoss EAP 6, there was no public API available for the cluster-wide HA singleton service. If you used the private **org.jboss.as.clustering.singleton.*** classes, you must change your code to use the new public **org.wildfly.clustering.singleton.*** packages when you migrate your application to JBoss EAP 7.

For more information about HA singleton services, see HA Singleton Service in the *Development Guide* for JBoss EAP. For information about HA singleton deployments, see HA Singleton Deployments in the *Development Guide* for JBoss EAP.

# CHAPTER 6. MISCELLANEOUS CHANGES

## 6.1. CHANGES TO DELIVERY OF JBOSS EAP NATIVES AND APACHE HTTP SERVER

JBoss EAP 7 natives are delivered differently in this release than in the past. Some now ship with the new Red Hat JBoss Core Services product, which is a set of supplementary software that is common to many of the Red Hat JBoss middleware products. The new product allows for faster distribution of updates and a more consistent update experience. The JBoss Core Services product is available for download in a different location on the Red Hat Customer Portal.

- The following table lists the differences in the delivery methods between the releases.

| Package | JBoss EAP 6 | JBoss EAP 7 |
|---|---|---|
| AIO Natives for Messaging | Delivered with the product in a separate "Native Utilities" download | Included within the JBoss EAP distribution. No additional download is required. |
| Apache HTTP Server | Delivered with the product in a separate "Apache HTTP Server" download | Delivered with the new JBoss Core Services product |
| mod_cluster, mod_jk, isapi, and nsapi connectors | Delivered with the product in a separate "Webserver Connector Natives" download | Delivered with the new JBoss Core Services product |
| JSVC | Delivered with the product in a separate "Native Utilities" download | Delivered with the new JBoss Core Services product |
| OpenSSL | Delivered with the product in a separate "Native Utilities" download | Delivered with the new JBoss Core Services product |
| tcnatives | Delivered with the product in a separate "Native Components" download | This was dropped in JBoss EAP 7 |

- You should also be aware of the following changes:
  - Support was dropped for mod_cluster and mod_jk connectors used with Apache HTTP Server from Red Hat Enterprise Linux RPM channels. If you run Apache HTTP Server from Red Hat Enterprise Linux RPM channels and need to configure load balancing for JBoss EAP 7 servers, you can do one of the following:
    - Use the Apache HTTP Server provided by JBoss Core Services.
    - You can configure JBoss EAP 7 to act as a front-end load balancer. For more information, see Configuring JBoss EAP as a Front-end Load Balancer in the JBoss EAP *Configuration Guide*.

  - You can deploy Apache HTTP Server on a machine that is supported and certified and then run the load balancer on that machine. For the list of supported configurations, see Overview of HTTP Connectors in the JBoss EAP 7 *Configuration Guide*.

- You can find more information about JBoss Core Services in the *Apache HTTP Server Installation Guide*.

## 6.2. CHANGES TO DEPLOYMENTS ON AMAZON EC2

A number of changes have been made to the Amazon Machine Images (AMI) in JBoss EAP 7. This section briefly summarizes some of those changes.

- The way you launch non-clustered and clustered JBoss EAP instances and domains in Amazon EC2 has changed significantly.
- JBoss EAP 6 used the **User Data:** field for JBoss EAP configuration. The AMI scripts that parsed the configuration in the **User Data:** field and started the servers automatically on instance startup have been removed from JBoss EAP 7.
- Red Hat JBoss Operations Network agent was installed in the previous release of JBoss EAP. In JBoss EAP 7, you must install it separately.

For details on deploying JBoss EAP 7 on Amazon EC2, see *Deploying JBoss EAP on Amazon Web Services*.

## 6.3. UNDEPLOYING APPLICATIONS THAT INCLUDE SHARED MODULES

Changes in the JBoss EAP 7.1 server and the Maven plug-in can result in the following failure when you attempt to undeploy your application. This error can occur if your application contains modules that interact with or depend on each other.

```
WFLYCTL0184:    New missing/unsatisfied dependencies
```

For example, assume you have an application that contains two Maven WAR project modules, **application-A** and **application-B**, that share data managed by the **data-sharing** module.

When you deploy this application, you must deploy the shared **data-sharing** module first, and then deploy the modules that depend on it. The deployment order is specified in the **<modules>** element of the parent **pom.xml** file. This is true in JBoss EAP 6.4 through JBoss EAP 7.2.

In releases prior to JBoss EAP 7.1, you could undeploy all of the archives for this application from the root of the parent project using the following command.

```
$ mvn wildfly:undeploy
```

In JBoss EAP 7.1 and later, you must first undeploy the archives that use the shared modules, and then undeploy the shared modules. Since there is no way to specify the order of undeployment in the project **pom.xml** file, you must undeploy the modules manually. You can accomplish this by running the following commands from the root of the parent directory.

```
$ mvn wildfly:undeploy -pl application-A,application-B
$ mvn wildfly:undeploy -pl data-shared
```

This new undeploy behavior is more correct and ensures that you do not end up in an unstable deployment state.

## 6.4. CHANGES TO JBOSS EAP SCRIPTS

The **add-user** script behavior has changed in JBoss EAP 7 due to a change in password policy. JBoss EAP 6 had a strict password policy. As a result, the **add-user** script rejected weak passwords that did not satisfy the minimum requirements. In JBoss EAP 7, weak passwords are accepted and a warning is issued. For more information, see Setting Add-User Utility Password Restrictions in the JBoss EAP *Configuration Guide*.

## 6.5. REMOVAL OF OSGI SUPPORT

When JBoss EAP 6.0 GA was first released, JBoss OSGi, an implementation of the OSGi specification, was included as a Technology Preview feature. With the release of JBoss EAP 6.1.0, JBoss OSGi was demoted from Technology Preview to Unsupported.

In JBoss EAP 6.1.0, the **configadmin** and **osgi** extension modules and subsystem configuration for a standalone server were moved to a separate ***EAP_HOME*/standalone/configuration/standalone-osgi.xml** configuration file. Because you should not migrate this unsupported configuration file, the removal of JBoss OSGi support should not impact the migration of a standalone server configuration. If you modified any of the other standalone configuration files to configure **osgi** or **configadmin**, those configurations must be removed.

For a managed domain, the **osgi** extension and subsystem configuration were removed from the ***EAP_HOME*/domain/configuration/domain.xml** file in the JBoss EAP 6.1.0 release. However, the **configadmin** module extension and subsystem configuration remain in the ***EAP_HOME*/domain/configuration/domain.xml** file. This configuration is no longer supported in JBoss EAP 7 and must be removed.

# CHAPTER 7. MIGRATING TO ELYTRON

## 7.1. OVERVIEW OF ELYTRON

JBoss EAP 7.1 introduced Elytron, which provides a single unified framework that can manage and configure access for both standalone servers and managed domains. It can also be used to configure security access for applications deployed to JBoss EAP servers.

### IMPORTANT

The architectures of Elytron and the legacy security subsystem that is based on PicketBox are very different. With Elytron, an attempt was made to create a solution that allows you to operate in the same security environments in which you currently operate; however, this does *not* mean that every PicketBox configuration option has an equivalent configuration option in Elytron.

If you are not able to find information in the documentation to help you achieve similar functionality using Elytron that you had when using the legacy security implementation, you can find help in one of the following ways.

- If you have a Red Hat Development subscription , you have access to Support Cases, Solutions, and Knowledge Articles on the Red Hat Customer Portal. You can also open a case with Technical Support and get help from the WildFly community as described below.

- If you do not have a Red Hat Development subscription, you can still access Knowledge Articles on the Red Hat Customer Portal. You can also join the user forums and live chat to ask questions of the WildFly community. The WildFly community offerings are actively monitored by the Elytron engineering team.

Your JBoss EAP 7.0 server configuration and deployments that use the legacy **security** subsystem, which is based on PicketBox, should run without changes on JBoss EAP 7.1 and later. PicketBox continues to support security domains, which allows applications to continue to use existing login modules. Security realms, which are used by the management layer for security, are also carried over and emulated by Elytron. This allows you to define authentication in both the **elytron** and legacy **security** subsystems and use them in parallel. For more information about how to configure your application to use Elytron and legacy security, see Configure Web Applications to Use Elytron or Legacy Security for Authentication in *How to Configure Identity Management* for JBoss EAP.

Even though PicketBox authentication continues to be supported, you are encouraged to switch to Elytron when you are ready to migrate your applications. One of the advantages for using Elytron security is that it provides a consistent security solution across the server and your applications. For information on how to migrate PicketBox authentication and authorization to use Elytron, see Migrate Authentication Configuration in this guide.

For an overview of the new resources that are available in the **elytron** subsystem, see Resources in the Elytron Subsystem in the JBoss EAP *Security Architecture* guide.

IMPORTANT

Be aware that if you do choose to use both the legacy **security** subsystem and Elytron in your deployments, invocations between deployments using different security architectures is not supported.

For more information about using these subsystems in parallel, see Using Elytron and Legacy Security Subsystems in Parallel in *How to Configure Identity Management* for JBoss EAP.

## 7.2. MIGRATE SECURE VAULTS AND PROPERTIES

### 7.2.1. Migrate Vaults to Secure Credential Storage

The vault that was used to store plain text string encryption in the legacy **security** subsystem in JBoss EAP 7.0 is not compatible with Elytron in JBoss EAP 7.1 or later, which uses a newly designed credential store to store strings. Credential stores safely encrypt credentials in a storage file outside of the JBoss EAP configuration files. You can use the implementation provided by Elytron or you can customize the configuration using the credential store APIs and SPIs. Each JBoss EAP server can contain multiple credential stores.

NOTE

If you previously used vault expressions to parameterize nonsensitive data, it is recommended that you replace the data with Elytron security properties.

If you continue to use the legacy **security** subsystem, you should not need to modify or update your vault data. However, if you plan to migrate your application to use Elytron, you must convert your existing vaults to credential stores so that they can be processed by the **elytron** subsystem. For more information about credential stores, see Credential Stores in *How to Configure Server Security* for JBoss EAP.

**Migrating Vault Data Using the WildFly Elytron Tool**

The WildFly Elytron Tool that ships with JBoss EAP provides a **vault** command to help you migrate vault content to credential stores. You execute the tool by running the **elytron-tool** script, which is located in the ***EAP_HOME*/bin** directory.

```
$ EAP_HOME/bin/elytron-tool.sh vault VAULT_ARGUMENTS
```

If you prefer, you can execute the tool by running the **java -jar** command.

```
$ java -jar EAP_HOME/bin/wildfly-elytron-tool.jar vault VAULT_ARGUMENTS
```

You can use the following command to get a description of all of the available arguments.

```
$ EAP_HOME/bin/elytron-tool.sh vault --help
```

NOTE

- The WildFly Elytron Tool cannot handle the first version of the security vault data files.
- You can enter the **--keystore-password** argument in masked format, as shown in the below example to migrate a single vault, or in clear text.
- The **--salt** and **--iteration** arguments are provided to supply information to decrypt the masked password or to generate a masked password in the output. If the **--salt** and **--iteration** arguments are omitted, default values are used.
- The **--summary** argument produces formatted management CLI commands that can be used to add the converted credential stores to the JBoss EAP configuration. Plain text passwords are masked in the summary output.

IMPORTANT

Be aware that credential stores can only be used for securing passwords. They do not support the vault expression feature that could be used anywhere in the management model.

Choose one of the following migration options:

- Migrate a Single Security Vault to a Credential Store
- Migrate Multiple Security Vaults to a Credential Store in Bulk

**Migrate a Single Security Vault to a Credential Store**
The following is an example of the command used to convert a single security vault to a credential store.

```
$ EAP_HOME/bin/elytron-tool.sh vault --enc-dir vault_data/ --keystore vault-jceks.keystore --
keystore-password MASK-2hKo56F1a3jYGnJwhPmiF5 --iteration 34 --salt 12345678 --alias test --
location cs-v1.store --summary
```

This command converts the security vault to a credential store and prints the summary of the management CLI commands that were used to convert it in the output.

```
Vault (enc-dir="vault_data/";keystore="vault-jceks.keystore") converted to credential store "cs-
v1.store"
Vault Conversion summary:
--------------------------------------
Vault Conversion Successful
CLI command to add new credential store:
/subsystem=elytron/credential-store=test:add(relative-
to=jboss.server.data.dir,create=true,modifiable=true,location="cs-v1.store",implementation-
properties={"keyStoreType"=>"JCEKS"},credential-reference={clear-text="MASK-
2hKo56F1a3jYGnJwhPmiF5;12345678;34"})
```

**Migrate Multiple Security Vaults to a Credential Store in Bulk**
You can convert multiple vaults to a credential store using the **--bulk-convert** argument and pointing to a bulk conversion descriptor file.

The examples in this section use the following bulk conversion descriptor file.

**Example: bulk-vault-conversion-descriptor.txt File**

```
keystore:vault-v1/vault-jceks.keystore
keystore-password:MASK-2hKo56F1a3jYGnJwhPmiF5
enc-dir:vault-v1/vault_data/
salt:12345678
iteration:34
location:v1-cs-1.store
alias:test

keystore:vault-v1/vault-jceks.keystore
keystore-password:secretsecret
enc-dir:vault-v1/vault_data/
location:v1-cs-2.store
alias:test

# different vault vault-v1-more
keystore:vault-v1-more/vault-jceks.keystore
keystore-password:MASK-2hKo56F1a3jYGnJwhPmiF5
enc-dir:vault-v1-more/vault_data/
salt:12345678
iteration:34
location:v1-cs-more.store
alias:test
```

A new conversion starts when each new **keystore:** line is encountered. All options are mandatory except for **salt**, **iteration**, and **properties**.

To perform the bulk conversion and generate output that formats the management CLI commands, execute the following command.

```
$ EAP_HOME/bin/elytron-tool.sh vault --bulk-convert path/to/bulk-vault-conversion-descriptor.txt --
summary
```

This command converts all of the security vaults specified in the file to a credential store and prints the summary of the management CLI commands that were used to convert them in the output.

```
Vault (enc-dir="vault-v1/vault_data/";keystore="vault-v1/vault-jceks.keystore") converted to credential
store "v1-cs-1.store"
Vault Conversion summary:
--------------------------------------
Vault Conversion Successful
CLI command to add new credential store:
/subsystem=elytron/credential-store=test:add(relative-
to=jboss.server.data.dir,create=true,modifiable=true,location="v1-cs-1.store",implementation-
properties={"keyStoreType"=>"JCEKS"},credential-reference={clear-text="MASK-
2hKo56F1a3jYGnJwhPmiF5;12345678;34"})
--------------------------------------

Vault (enc-dir="vault-v1/vault_data/";keystore="vault-v1/vault-jceks.keystore") converted to credential
store "v1-cs-2.store"
Vault Conversion summary:
--------------------------------------
Vault Conversion Successful
CLI command to add new credential store:
```

```
/subsystem=elytron/credential-store=test:add(relative-
to=jboss.server.data.dir,create=true,modifiable=true,location="v1-cs-2.store",implementation-
properties={"keyStoreType"=>"JCEKS"},credential-reference={clear-text="secretsecret"})
--------------------------------------

Vault (enc-dir="vault-v1-more/vault_data/";keystore="vault-v1-more/vault-jceks.keystore") converted
to credential store "v1-cs-more.store"
Vault Conversion summary:
--------------------------------------
Vault Conversion Successful
CLI command to add new credential store:
/subsystem=elytron/credential-store=test:add(relative-
to=jboss.server.data.dir,create=true,modifiable=true,location="v1-cs-more.store",implementation-
properties={"keyStoreType"=>"JCEKS"},credential-reference={clear-text="MASK-
2hKo56F1a3jYGnJwhPmiF5;12345678;34"})
--------------------------------------
```

### 7.2.2. Migrate Security Properties to Elytron

The examples in this section assume that the **group.name** and **encoding.algorithm** security properties are defined as **security-properties** in the legacy **security** subsystem as follows.

#### Example: Security Properties Defined in the security Subsystem

```
<subsystem xmlns="urn:jboss:domain:security:2.0">
    ...
    <security-properties>
        <property name="group.name" value="engineering-group" />
        <property name="encoding.algorithm" value="BASE64" />
    </security-properties>
</subsystem>
```

To define the same security properties in the **elytron** subsystem, set the **security-properties** attribute of the **elytron** subsystem using the following management CLI command.

```
/subsystem=elytron:write-attribute(name=security-properties, value={ group.name = "engineering-
group", encoding.algorithm = "BASE64" })
```

This configures the following **security-properties** in the **elytron** subsystem in the server configuration file.

```
<subsystem xmlns="urn:wildfly:elytron:4.0" final-providers="combined-providers" disallowed-
providers="OracleUcrypto">
    <security-properties>
        <security-property name="group.name" value="engineering-group"/>
        <security-property name="encoding.algorithm" value="BASE64"/>
    </security-properties>
    ...
</subsystem>
```

The **write-attribute** operation used in the previous command overwrites the existing properties. To add or change a security property without impacting other security properties, use the **map** operation in the management CLI command.

```
/subsystem=elytron:map-put(name=security-properties, key=group.name, value=technical-support)
```

In a similar manner, you can remove a specific security property by using the **map-remove** operation.

```
/subsystem=elytron:map-remove(name=security-properties, key=group.name)
```

## 7.3. MIGRATE AUTHENTICATION CONFIGURATION

### 7.3.1. Migrate Properties-based Authentication and Authorization to Elytron

#### 7.3.1.1. Migrate PicketBox Properties-based Configuration to Elytron

This section describes how to migrate PicketBox properties-based authentication to Elytron. You can choose to partially migrate properties-based authentication by only exposing the PicketBox security domain to Elytron or you can fully migrate the properties-based authentication configurations to use Elytron.

The following procedures assume that the deployed web application you plan to migrate is configured to require form-based authentication. The application is referencing a PicketBox security domain and is using the **UsersRolesLoginModule** to load user information from the **example-users.properties** and **example-roles.properties** files. These examples also assume that the security domain is defined in the legacy **security** subsystem using the following management CLI commands.

**Example: PicketBox Properties-based Configuration Commands**

```
/subsystem=security/security-domain=application-security:add
/subsystem=security/security-domain=application-security/authentication=classic:add(login-modules=
[{code=UsersRoles, flag=Required, module-options=
{usersProperties=file://${jboss.server.config.dir}/example-users.properties,
rolesProperties=file://${jboss.server.config.dir}/example-roles.properties}}])
```

This results in the following server configuration.

**Example: PicketBox Properties-based Security Domain Configuration**

```
<security-domain name="application-security">
  <authentication>
    <login-module code="UsersRoles" flag="required">
      <module-option name="usersProperties" value="file://${jboss.server.config.dir}/example-
users.properties"/>
      <module-option name="rolesProperties" value="file://${jboss.server.config.dir}/example-
roles.properties"/>
    </login-module>
  </authentication>
</security-domain>
```

Choose one of the following migration options:

- Partially Migrate by Exposing the PicketBox Security Domain to Elytron .
- Fully Migrate Properties-based Authentication to Elytron

**Partially Migrate by Exposing the PicketBox Security Domain to Elytron**

You can expose a PicketBox security domain as an Elytron security realm so that it can be wired into an Elytron configuration; however, doing so creates a dependency on the legacy **security** subsystem. If you are only migrating properties-based authentication, it is recommended that you fully migrate the application to Elytron to avoid the unnecessary dependency on the legacy **security** subsystem. However, a partial migration can be an intermediate solution when it is not possible to fully migrate the application to use Elytron.

Follow this procedure to add an existing PicketBox security realm configuration as an Elytron security realm.

1. Add a mapping to the Elytron security realm within the legacy **security** subsystem.

   ```
   /subsystem=security/elytron-realm=application-security:add(legacy-jaas-config=application-security)
   ```

   This configures the following Elytron security realm in the **security** subsystem of the server configuration file.

   ```
   <subsystem xmlns="urn:jboss:domain:security:2.0">
     ...
     <elytron-integration>
       <security-realms>
         <elytron-realm name="application-security" legacy-jaas-config="application-security"/>
       </security-realms>
     </elytron-integration>
     ...
   </subsystem>
   ```

2. Define a security domain in the **elytron** subsystem that references the exported security realm.

   ```
   /subsystem=elytron/security-domain=application-security:add(realms=[{realm=application-security}], default-realm=application-security, permission-mapper=default-permission-mapper)
   ```

   This results in the following **elytron** subsystem configuration in the server configuration file.

   ```
   <subsystem xmlns="urn:wildfly:elytron:4.0" final-providers="combined-providers" disallowed-providers="OracleUcrypto">
     ...
     <security-domains>
       ...
       <security-domain name="application-security" default-realm="application-security"
   permission-mapper="default-permission-mapper">
         <realm name="application-security"/>
       </security-domain>
     </security-domains>
     ...
   </subsystem>
   ```

3. In the **undertow** subsystem, map the application security domain referenced by the deployment to the newly defined security domain.

   ```
   /subsystem=undertow/application-security-domain=application-security:add(security-domain=application-security)
   ```

This results in the following **undertow** subsystem configuration in the server configuration file.

```
<subsystem xmlns="urn:wildfly:elytron:4.0">
  ...
  <application-security-domains>
    <application-security-domain name="application-security" security-domain="application-
security"/>
  </application-security-domains>
  ...
</subsystem>
```

**NOTE**

If the application was already deployed prior to this configuration, you must reload the server or redeploy the application for the new application security domain mapping to take effect.

4. Verify the mapping was applied to the deployment using the following management CLI command. The deployment used in this example is **HelloWorld.war**. The output from the this command shows this deployment is referencing the Elytron mapping.

```
/subsystem=undertow/application-security-domain=application-security:read-
resource(include-runtime=true)

{
"outcome" => "success",
   "result" => {
      "enable-jacc" => false,
      "http-authentication-factory" => undefined,
      "override-deployment-config" => false,
      "referencing-deployments" => ["HelloWorld.war"],
      "security-domain" => "application-security",
      "setting" => undefined
   }
}
```

At this stage, the previously defined security domain is used for its **LoginModule** configuration, but it is wrapped by Elytron components, which take over authentication.

### Fully Migrate Properties-based Authentication to Elytron

Follow these steps to fully migrate the PicketBox properties-based authentication to Elytron. This procedure assumes you are starting with the legacy configuration described in the introduction to this section and have *not* migrated to the partially migrated solution. When you have complete this process, any security domain definition that exists in the legacy **security** subsystem remains completely independent from the Elytron configuration.

1. Define a new security realm in the **elytron** subsystem that references the PicketBox properties files.

```
/subsystem=elytron/properties-realm=application-properties:add(users-properties=
{path=example-users.properties, relative-to=jboss.server.config.dir, plain-text=true, digest-
realm-name="Application Security"}, groups-properties={path=example-roles.properties,
relative-to=jboss.server.config.dir}, groups-attribute=Roles)
```

2. Define a security domain subsystem in the **elytron** subsystem.

```
/subsystem=elytron/security-domain=application-security:add(realms=[{realm=application-
properties}], default-realm=application-properties, permission-mapper=default-permission-
mapper)
```

This results in the following **elytron** subsystem configuration in the server configuration file.

```
<subsystem xmlns="urn:wildfly:elytron:4.0" final-providers="combined-providers" disallowed-
providers="OracleUcrypto">
  ...
  <security-domains>
    ...
    <security-domain name="application-security" default-realm="application-properties"
permission-mapper="default-permission-mapper">
      <realm name="application-properties"/>
    </security-domain>
  </security-domains>
  <security-realms>
    ...
    <properties-realm name="application-properties" groups-attribute="Roles">
      <users-properties path="example-users.properties" relative-to="jboss.server.config.dir"
digest-realm-name="Application Security" plain-text="true"/>
      <groups-properties path="example-roles.properties" relative-to="jboss.server.config.dir"/>
    </properties-realm>
  </security-realms>
  ...
</subsystem>
```

3. Map the application security domain referenced by the deployment to the newly defined HTTP authentication factory in the **undertow** subsystem.

```
/subsystem=undertow/application-security-domain=application-security:add(security-
domain=application-security)
```

This results in the following **undertow** subsystem configuration in the server configuration file.

```
<subsystem xmlns="urn:jboss:domain:undertow:7.0">
  ...
  <application-security-domains>
    <application-security-domain name="application-security" security-domain="application-
security"/>
  </application-security-domains>
  ...
</subsystem>
```

4. You must reload the server or redeploy the application for the new application security domain mapping to take effect.

Authentication is now configured to be equivalent to the PicketBox configuration; however Elytron components are now used exclusively for authentication.

#### 7.3.1.2. Migrate Legacy Properties-based Configuration to Elytron

This section describes how to migrate a legacy security realm that loads user, password, and group information from properties files to Elytron. This type of legacy security realm is typically used to secure either the management interfaces or remoting connectors.

These examples assume that the legacy security domain is defined using the following management CLI commands.

### Example: Legacy Security Realm Commands

```
/core-service=management/security-realm=ApplicationSecurity:add
/core-service=management/security-
realm=ApplicationSecurity/authentication=properties:add(relative-to=jboss.server.config.dir,
path=example-users.properties, plain-text=true)
/core-service=management/security-realm=ApplicationSecurity/authorization=properties:add(relative-
to=jboss.server.config.dir, path=example-roles.properties)
```

This results in the following server configuration.

### Example: Legacy Security Realm Configuration

```
<security-realm name="ApplicationSecurity">
  <authentication>
    <properties path="example-users.properties" relative-to="jboss.server.config.dir" plain-text="true"/>
  </authentication>
  <authorization>
    <properties path="example-roles.properties" relative-to="jboss.server.config.dir"/>
  </authorization>
</security-realm>
```

One of the motivations for adding the Elytron security to the application server is to allow a consistent security solution to be used across the server. The initial steps to migrate a properties-based legacy security realm to Elytron are similar to those used to migrate a PicketBox properties-based authentication to Elytron. Follow these steps to migrate a properties-based legacy security realm to Elytron.

1. Define a new security realm in the **elytron** subsystem that references the properties files.

   ```
   /subsystem=elytron/properties-realm=application-properties:add(users-properties=
   {path=example-users.properties, relative-to=jboss.server.config.dir, plain-text=true, digest-
   realm-name="Application Security"}, groups-properties={path=example-roles.properties,
   relative-to=jboss.server.config.dir}, groups-attribute=Roles)
   ```

2. Define a security domain subsystem in the **elytron** subsystem.

   ```
   /subsystem=elytron/security-domain=application-security:add(realms=[{realm=application-
   properties}], default-realm=application-properties, permission-mapper=default-permission-
   mapper)
   ```

   This results in the following Elytron configuration.

   ```
   <subsystem xmlns="urn:wildfly:elytron:4.0" final-providers="combined-providers" disallowed-
   providers="OracleUcrypto">
     ...
     <security-domains>
   ```

```
    ...
    <security-domain name="application-security" default-realm="application-properties"
permission-mapper="default-permission-mapper">
      <realm name="application-properties"/>
    </security-domain>
  </security-domains>
  <security-realms>
    ...
    <properties-realm name="application-properties" groups-attribute="Roles">
      <users-properties path="example-users.properties" relative-to="jboss.server.config.dir"
digest-realm-name="Application Security" plain-text="true"/>
      <groups-properties path="example-roles.properties" relative-to="jboss.server.config.dir"/>
    </properties-realm>
  </security-realms>
    ...
</subsystem>
```

3. Define a **sasl-authentication-factory** so that the legacy security realm can also be used for Simple Authentication Security Layer (SASL) authentication.

```
/subsystem=elytron/sasl-authentication-factory=application-security-sasl:add(sasl-server-
factory=elytron, security-domain=application-security, mechanism-configurations=
[{mechanism-name=PLAIN}])
```

This results in the following Elytron configuration.

```
<subsystem xmlns="urn:wildfly:elytron:4.0" final-providers="combined-providers" disallowed-
providers="OracleUcrypto">
  ...
  <sasl>
    ...
    <sasl-authentication-factory name="application-security-sasl" sasl-server-factory="elytron"
security-domain="application-security">
      <mechanism-configuration>
        <mechanism mechanism-name="PLAIN"/>
      </mechanism-configuration>
    </sasl-authentication-factory>
    ...
  </sasl>
</subsystem>
```

4. Configure a remoting connector for the SASL authentication and remove the association with the legacy security realm.

```
/subsystem=remoting/http-connector=http-remoting-connector:write-attribute(name=sasl-
authentication-factory, value=application-security-sasl)
/subsystem=remoting/http-connector=http-remoting-connector:undefine-
attribute(name=security-realm)
```

This results in the following configuration in the **remoting** subsystem of the server configuration file.

```
<subsystem xmlns="urn:jboss:domain:remoting:4.0">
  ...
  <http-connector name="http-remoting-connector" connector-ref="default" sasl-
authentication-factory="application-security-sasl"/>
</subsystem>
```

5. Add the two authentication factories and remove the legacy security realm references to secure the **http-interface** with Elytron.

   ```
   /core-service=management/management-interface=http-interface:write-attribute(name=http-
   authentication-factory, value=application-security-http)
   /core-service=management/management-interface=http-interface:write-attribute(name=http-
   upgrade.sasl-authentication-factory, value=application-security-sasl)
   /core-service=management/management-interface=http-interface:undefine-
   attribute(name=security-realm)
   ```

   This results in the following configuration.

   ```
   <management-interfaces>
     <http-interface http-authentication-factory="application-security-http">
       <http-upgrade enabled="true" sasl-authentication-factory="application-security-sasl"/>
       <socket-binding http="management-http"/>
     </http-interface>
   </management-interfaces>
   ```

   NOTE

   You should choose more suitable names than those used in these examples when securing management interfaces.

The migration of the legacy properties-based configuration to Elytron is now complete.

### 7.3.2. Migrate LDAP Authentication Configuration to Elytron

This section describes how to migrate legacy LDAP authentication to Elytron so that it can manage the information as identity attributes. Much of the information provided in the section entitled Migrate Properties-based Authentication and Authorization to Elytron applies here, particularly regarding how to define security domains and authentication factories, and how to map them to be used for authentication. This section does not repeat those instructions, so be sure to read through that section before you continue.

The following examples assume that group or role information is loaded directly from LDAP and that the legacy LDAP authentication is configured as follows.

- The LDAP server contains the following user and group entries.

  **Example: LDAP Server User Entries**

  ```
  dn: uid=TestUserOne,ou=users,dc=group-to-principal,dc=wildfly,dc=org
  objectClass: top
  objectClass: inetOrgPerson
  objectClass: uidObject
  objectClass: person
  objectClass: organizationalPerson
  cn: Test User One
  ```

```
sn: Test User One
uid: TestUserOne
userPassword: {SSHA}UG8ov2rnrnBKakcARVvraZHqTa7mFWJZlWt2HA==
```

**Example: LDAP Server Group Entries**

```
dn: uid=GroupOne,ou=groups,dc=group-to-principal,dc=wildfly,dc=org
objectClass: top
objectClass: groupOfUniqueNames
objectClass: uidObject
cn: Group One
uid: GroupOne
uniqueMember: uid=TestUserOne,ou=users,dc=group-to-principal,dc=wildfly,dc=org
```

For authentication purposes the user name is matched against the **uid** attribute and the resulting group name is taken from the **uid** attribute of the group entry.

- The connection to the LDAP server and related security realm is defined using the following management CLI commands.

**Example: LDAP Security Realm Configuration Commands**

```
batch
/core-service=management/ldap-
connection=MyLdapConnection:add(url="ldap://localhost:10389", search-
dn="uid=admin,ou=system", search-credential="secret")

/core-service=management/security-realm=LDAPRealm:add
/core-service=management/security-
realm=LDAPRealm/authentication=ldap:add(connection="MyLdapConnection", username-
attribute=uid, base-dn="ou=users,dc=group-to-principal,dc=wildfly,dc=org")

/core-service=management/security-
realm=LDAPRealm/authorization=ldap:add(connection=MyLdapConnection)
/core-service=management/security-realm=LDAPRealm/authorization=ldap/username-to-
dn=username-filter:add(attribute=uid, base-dn="ou=users,dc=group-to-
principal,dc=wildfly,dc=org")
/core-service=management/security-realm=LDAPRealm/authorization=ldap/group-
search=group-to-principal:add(base-dn="ou=groups,dc=group-to-
principal,dc=wildfly,dc=org", iterative=true, prefer-original-connection=true, principal-
attribute=uniqueMember, search-by=DISTINGUISHED_NAME, group-name=SIMPLE,
group-name-attribute=uid)
run-batch
```

This results in the following server configuration.

**Example: LDAP Security Realm Configuration**

```
<management>
  <security-realms>
    ...
    <security-realm name="LDAPRealm">
      <authentication>
        <ldap connection="MyLdapConnection" base-dn="ou=users,dc=group-to-
```

```
principal,dc=wildfly,dc=org">
          <username-filter attribute="uid"/>
        </ldap>
      </authentication>
      <authorization>
        <ldap connection="MyLdapConnection">
          <username-to-dn>
            <username-filter base-dn="ou=users,dc=group-to-principal,dc=wildfly,dc=org"
attribute="uid"/>
          </username-to-dn>
          <group-search group-name="SIMPLE" iterative="true" group-name-attribute="uid">
            <group-to-principal search-by="DISTINGUISHED_NAME" base-
dn="ou=groups,dc=group-to-principal,dc=wildfly,dc=org" prefer-original-connection="true">
              <membership-filter principal-attribute="uniqueMember"/>
            </group-to-principal>
          </group-search>
        </ldap>
      </authorization>
    </security-realm>
  </security-realms>
  <outbound-connections>
    <ldap name="MyLdapConnection" url="ldap://localhost:10389" search-
dn="uid=admin,ou=system" search-credential="secret"/>
  </outbound-connections>
  ...
</management>
```

- The following management CLI commands are used to configure a PicketBox security domain, which uses the **LdapExtLoginModule** to verify a user name and password.

  **Example: Security Domain Configuration Commands**

```
/subsystem=security/security-domain=application-security:add
/subsystem=security/security-domain=application-security/authentication=classic:add(login-
modules=[{code=LdapExtended, flag=Required, module-options={
java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory,
java.naming.provider.url=ldap://localhost:10389, java.naming.security.authentication=simple,
bindDN="uid=admin,ou=system", bindCredential=secret, baseCtxDN="ou=users,dc=group-
to-principal,dc=wildfly,dc=org", baseFilter="(uid={0})", rolesCtxDN="ou=groups,dc=group-to-
principal,dc=wildfly,dc=org", roleFilter="(uniqueMember={1})", roleAttributeID="uid" }}])
```

  This results in the following server configuration.

  **Example: Security Domain Configuration**

```
<subsystem xmlns="urn:jboss:domain:security:2.0">
  ...
  <security-domains>
    ...
    <security-domain name="application-security">
      <authentication>
        <login-module code="LdapExtended" flag="required">
          <module-option name="java.naming.factory.initial"
value="com.sun.jndi.ldap.LdapCtxFactory"/>
          <module-option name="java.naming.provider.url" value="ldap://localhost:10389"/>
          <module-option name="java.naming.security.authentication" value="simple"/>
          <module-option name="bindDN" value="uid=admin,ou=system"/>
          <module-option name="bindCredential" value="secret"/>
          <module-option name="baseCtxDN" value="ou=users,dc=group-to-
principal,dc=wildfly,dc=org"/>
          <module-option name="baseFilter" value="(uid={0})"/>
          <module-option name="rolesCtxDN" value="ou=groups,dc=group-to-
principal,dc=wildfly,dc=org"/>
          <module-option name="roleFilter" value="(uniqueMember={1})"/>
          <module-option name="roleAttributeID" value="uid"/>
        </login-module>
      </authentication>
    </security-domain>
  </security-domains>
</subsystem>
```

#### 7.3.2.1. Migrate the Legacy LDAP Authentication to Elytron

Follow these steps to migrate the previous LDAP authentication example configuration to Elytron. This section applies to the migration of a legacy security LDAP realm as well as a PicketBox LDAP security domain.

1. Define a connection to LDAP in the **elytron** subsystem.

```
/subsystem=elytron/dir-context=ldap-connection:add(url=ldap://localhost:10389,
principal="uid=admin, ou=system", credential-reference={clear-text=secret})
```

2. Create a security realm to search LDAP and verify the supplied password.

```
/subsystem=elytron/ldap-realm=ldap-realm:add(dir-context=ldap-connection, direct-
verification=true, identity-mapping={search-base-dn="ou=users, dc=group-to-principal,
dc=wildfly, dc=org", rdn-identifier="uid", attribute-mapping=[{filter-base-dn="ou=groups,
dc=group-to-principal, dc=wildfly, dc=org", filter="(uniqueMember={1})", from="uid",
to="Roles"}]})
```

These steps result in the following **elytron** subsystem configuration in the server configuration file.

```
<subsystem xmlns="urn:wildfly:elytron:4.0" final-providers="combined-providers" disallowed-
providers="OracleUcrypto">
  ...
  <security-realms>
    ...
    <ldap-realm name="ldap-realm" dir-context="ldap-connection" direct-verification="true">
      <identity-mapping rdn-identifier="uid" search-base-dn="ou=users,dc=group-to-
principal,dc=wildfly,dc=org">
        <attribute-mapping>
          <attribute from="uid" to="Roles" filter="(uniqueMember={1})" filter-base-
dn="ou=groups,dc=group-to-principal,dc=wildfly,dc=org"/>
        </attribute-mapping>
      </identity-mapping>
    </ldap-realm>
  </security-realms>
  ...
  <dir-contexts>
```

```
    <dir-context name="ldap-connection" url="ldap://localhost:10389"
principal="uid=admin,ou=system">
      <credential-reference clear-text="secret"/>
    </dir-context>
  </dir-contexts>
</subsystem>
```

NOTE

By default, if no **role-decoder** is defined for a given **security-domain**, the "Roles" identity attribute is mapped to the identity roles.

Information loaded from LDAP can now be associated with identities as attributes. These attributes can be mapped to roles, but they can also be loaded and used for other purposes. The newly created security realm can be used in a security domain in the same way as it is described in the Migrate Properties-based Authentication and Authorization to Elytron section of this guide.

### 7.3.3. Migrate Database Authentication Configuration to Elytron

This section describes how to migrate JDBC datasource-based PicketBox authentication to Elytron. Much of the information provided in the section entitled Migrate Properties-based Authentication and Authorization to Elytron applies here, particularly regarding how to define security domains and authentication factories, and how to map them to be used for authentication. This section does not repeat those instructions, so be sure to read through that section before you continue.

The following examples assume that the user authentication data is stored in a database table created using syntax similar to the following example.

#### Example: Syntax to Create the Database User Table

```
CREATE TABLE User (
    id BIGINT NOT NULL,
    username VARCHAR(255),
    password VARCHAR(255),
    role ENUM('admin', 'manager', 'user'),
    PRIMARY KEY (id),
    UNIQUE (username)
)
```

For authentication purposes the username is matched against data stored in the **username** column, the password is expected to be stored as a hex-encoded MD5 hash in the **password** column, and the user role for authorization purposes is stored in the **role** column.

The PicketBox security domain is configured to use a JBDC datasource to retrieve data from the database table, and then use it to verify the username and password, and to assign roles. Assume the PicketBox security domain is configured using the following management CLI commands.

#### Example: PicketBox Database LoginModule Configuration Commands

```
/subsystem=security/security-domain=application-security:add
/subsystem=security/security-domain=application-security/authentication=classic:add( login-
modules=[ { code=Database, flag=Required, module-options={
```

```
dsJndiName="java:jboss/datasources/ExampleDS", principalsQuery="SELECT password FROM
User WHERE username = ?", rolesQuery="SELECT role, 'Roles' FROM User WHERE username =
?", hashAlgorithm=MD5, hashEncoding=base64 } } ] )
```

This results in the following **login-module** configuration in the legacy **security** subsystem.

### Example: PicketBox LoginModule Configuration

```
<subsystem xmlns="urn:jboss:domain:security:2.0">
  <security-domains>
    ...
    <security-domain name="application-security">
      <authentication>
        <login-module code="Database" flag="required">
          <module-option name="dsJndiName" value="java:jboss/datasources/ExampleDS"/>
          <module-option name="principalsQuery" value="SELECT password FROM User WHERE
username = ?"/>
          <module-option name="rolesQuery" value="SELECT role, 'Roles' FROM User WHERE
username = ?"/>
          <module-option name="hashAlgorithm" value="MD5"/>
          <module-option name="hashEncoding" value="base64"/>
        </login-module>
      </authentication>
    </security-domain>
  </security-domains>
</subsystem>
```

#### 7.3.3.1. Migrate the Legacy Database Authentication to Elytron

To migrate the previous database authentication example configuration to Elytron, you must define a JDBC realm to enable JDBC datasource access by Elytron.

Use the following management command to define the **jdbc-realm**.

```
/subsystem=elytron/jdbc-realm=jdbc-realm:add(principal-query=[ { data-source=ExampleDS,
sql="SELECT role, password FROM User WHERE username = ?", attribute-mapping=[{index=1,
to=Roles } ] simple-digest-mapper={algorithm=simple-digest-md5, password-index=2} } ] )
```

This results in the following **jdbc-realm** configuration in the **elytron** subsystem of the server configuration file.

```
<subsystem xmlns="urn:wildfly:elytron:4.0" final-providers="combined-providers" disallowed-
providers="OracleUcrypto">
  ...
  <security-realms>
    ...
    <jdbc-realm name="jdbc-realm">
      <principal-query sql="SELECT role, password FROM User WHERE username = ?" data-
source="ExampleDS">
        <attribute-mapping>
          <attribute to="Roles" index="1"/>
        </attribute-mapping>
        <simple-digest-mapper password-index="2"/>
      </principal-query>
    </jdbc-realm>
    ...
  </security-realms>
  ...
</subsystem>
```

Elytron now manages the database authentication using the JDBC realm configuration. Elytron is more efficient than PicketBox because it uses one SQL query to obtain all of the user attributes and credentials, and then extracts data from the SQL results and creates a mapping of the attributes to use for authentication.

### 7.3.4. Migrate Kerberos Authentication to Elytron

When working with a Kerberos configuration, the JBoss EAP server can rely on configuration information from the environment, or the key configuration can be specified using system properties. This section discusses how to migrate Kerberos HTTP and Kerberos SASL authentication.

The examples that follow assume that Kerberos is configured using the following system properties. These system properties are applicable to both the legacy configuration and the migrated Elytron configuration.

**Example: Kerberos System Properties Management CLI Commands**

```
# Enable debugging
/system-property=sun.security.krb5.debug:add(value=true)
# Identify the Kerberos realm to use
/system-property=java.security.krb5.realm:add(value=ELYTRON.ORG)
# Identify the address of the KDC
/system-property=java.security.krb5.kdc:add(value=kdc.elytron.org)
```

**Example: Kerberos System Properties Server Configuration**

```
<system-properties>
  <property name="sun.security.krb5.debug" value="true"/>
  <property name="java.security.krb5.realm" value="ELYTRON.ORG"/>
  <property name="java.security.krb5.kdc" value="kdc.elytron.org"/>
</system-properties>
```

Choose one of the following migration options:

- Migrate Kerberos HTTP Authentication.
- Migrate Kerberos Remoting SASL Authentication.

**Migrate Kerberos HTTP Authentication**

In legacy security configurations, you can define a security realm to enable SPNEGO authentication for the HTTP management interface as follows.

**Example: Enable SPNEGO authentication for the HTTP management interface**

```
/core-service=management/security-realm=Kerberos:add
/core-service=management/security-realm=Kerberos/server-identity=kerberos:add
/core-service=management/security-realm=Kerberos/server-identity=kerberos/keytab=HTTP\/test-
```

```
server.elytron.org@ELYTRON.ORG:add(path=/path/to/test-server.keytab, debug=true)
/core-service=management/security-realm=Kerberos/authentication=kerberos:add(remove-
realm=true)
```

**Example: Kerberos Security Realm Configuration**

```
<security-realms>
  ...
  <security-realm name="Kerberos">
    <server-identities>
      <kerberos>
        <keytab principal="HTTP/test-server.elytron.org@ELYTRON.ORG" path="/path/to/test-
server.keytab" debug="true"/>
      </kerberos>
    </server-identities>
    <authentication>
      <kerberos remove-realm="true"/>
    </authentication>
  </security-realm>
</security-realms>
```

You can also define a pair of legacy security domains to allow applications to use Kerberos HTTP authentication.

**Example: Define Multiple Security Domains**

```
# Define the first security domain
/subsystem=security/security-domain=host:add
/subsystem=security/security-domain=host/authentication=classic:add
/subsystem=security/security-domain=host/authentication=classic/login-
module=1:add(code=Kerberos, flag=Required, module-options={storeKey=true, useKeyTab=true,
principal=HTTP/test-server.elytron.org@ELYTRON.ORG, keyTab=path/to/test-server.keytab,
debug=true}

# Define the second SPNEGO security domain
/subsystem=security/security-domain=SPNEGO:add
/subsystem=security/security-domain=SPNEGO/authentication=classic:add
/subsystem=security/security-domain=SPNEGO/authentication=classic/login-
module=1:add(code=SPNEGO, flag=requisite,  module-options={password-stacking=useFirstPass,
serverSecurityDomain=host})
/subsystem=security/security-domain=SPNEGO/authentication=classic/login-module=1:write-
attribute(name=module, value=org.jboss.security.negotiation)
/subsystem=security/security-domain=SPNEGO/authentication=classic/login-
module=2:add(code=UsersRoles, flag=required, module-options={password-stacking=useFirstPass,
usersProperties=  /path/to/kerberos/spnego-users.properties, rolesProperties=
/path/to/kerberos/spnego-roles.properties, defaultUsersProperties=  /path/to/kerberos/spnego-
users.properties, defaultRolesProperties=  /path/to/kerberos/spnego-roles.properties})
```

**Example: Configuration Using a Pair of Security Domains**

```
<subsystem xmlns="urn:jboss:domain:security:2.0">
  <security-domains>
    ...
    <security-domain name="host">
```

```
      <authentication>
        <login-module name="1" code="Kerberos" flag="required">
          <module-option name="storeKey" value="true"/>
          <module-option name="useKeyTab" value="true"/>
          <module-option name="principal" value="HTTP/test-server.elytron.org@ELYTRON.ORG"/>
          <module-option name="keyTab" value="/path/to/test-server.keytab"/>
          <module-option name="debug" value="true"/>
        </login-module>
      </authentication>
    </security-domain>
    <security-domain name="SPNEGO">
      <authentication>
        <login-module name="1" code="SPNEGO" flag="requisite"
module="org.jboss.security.negotiation">
          <module-option name="password-stacking" value="useFirstPass"/>
          <module-option name="serverSecurityDomain" value="host"/>
        </login-module>
        <login-module name="2" code="UsersRoles" flag="required">
          <module-option name="password-stacking" value="useFirstPass"/>
          <module-option name="usersProperties" value="path/to/kerberos/spnego-users.properties"/>
          <module-option name="rolesProperties" value="  /path/to/kerberos/spnego-roles.properties"/>
          <module-option name="defaultUsersProperties" value="  /path/to/kerberos/spnego-
users.properties"/>
          <module-option name="defaultRolesProperties" value="  /path/to/kerberos/spnego-
roles.properties"/>
        </login-module>
      </authentication>
    </security-domain>
  </security-domains>
</subsystem>
```

The legacy applications are then deployed referencing the SPNEGO security domain and secured with the SPNEGO mechanism.

**Migrate the Kerberos HTTP Authentication to Elytron**

Both the management interface and applications can be secured in Elytron by using a security realm and a Kerberos security factory.

1. Define a security realm to be used to load identity information.

```
/subsystem=elytron/properties-realm=spnego-properties:add(users-properties=
{path=path/to/spnego-users.properties, plain-text=true, digest-realm-
name=ELYTRON.ORG}, groups-properties={path=path/to/spnego-roles.properties})
```

2. Define a Kerberos security factory that allows the server to load its own Kerberos identity.

```
/subsystem=elytron/kerberos-security-factory=test-server:add(path=path/to/test-
server.keytab, principal=HTTP/test-server.elytron.org@ELYTRON.ORG, debug=true)
```

3. Define a security domain to pull together the policy as well as an HTTP authentication factory for the authentication policy.

```
/subsystem=elytron/security-domain=SPNEGODomain:add(default-realm=spnego-
properties, realms=[{realm=spnego-properties, role-decoder=groups-to-roles}], permission-
mapper=default-permission-mapper)
```

```
/subsystem=elytron/http-authentication-factory=spnego-http-authentication:add(security-
domain=SPNEGODomain, http-server-mechanism-factory=global,mechanism-
configurations=[{mechanism-name=SPNEGO, credential-security-factory=test-server}])
```

This results in the following configuration in the **elytron** subsystem of the server configuration file.

**Example: Migrated Elytron Configuration**

```
<subsystem xmlns="urn:wildfly:elytron:4.0" final-providers="combined-providers" disallowed-
providers="OracleUcrypto">
  ...
  <security-domains>
  ...
    <security-domain name="SPNEGODomain" default-realm="spnego-properties"
permission-mapper="default-permission-mapper">
      <realm name="spnego-properties" role-decoder="groups-to-roles"/>
    </security-domain>
  </security-domains>
  <security-realms>
    ...
    <properties-realm name="spnego-properties">
      <users-properties path="path/to/spnego-users.properties" digest-realm-
name="ELYTRON.ORG" plain-text="true"/>
      <groups-properties path="path/to/spnego-roles.properties"/>
    </properties-realm>
  </security-realms>
  <credential-security-factories>
    <kerberos-security-factory name="test-server" principal="HTTP/test-
server.elytron.org@ELYTRON.ORG" path="path/to/test-server.keytab" debug="true"/>
  </credential-security-factories>
  ...
  <http>
    ...
    <http-authentication-factory name="spnego-http-authentication" http-server-mechanism-
factory="global" security-domain="SPNEGODomain">
      <mechanism-configuration>
        <mechanism mechanism-name="SPNEGO" credential-security-factory="test-server"/>
      </mechanism-configuration>
    </http-authentication-factory>
    ...
  </http>
  ...
</subsystem>
```

4. To secure the application, define an application security domain in the **undertow** subsystem to map security domains to this **http-authentication-factory**. The HTTP management interface can be updated to reference the **http-authentication-factory** defined in this configuration. This process is documented in the Migrate Properties-based Authentication and Authorization to Elytron section of this guide.

**Migrate Kerberos Remoting SASL Authentication**

It is possible to define a legacy security realm for Kerberos / GSSAPI SASL authentication to be used for remoting authentication, such as the native management interface.

**Example: Kerberos Authentication for Remoting Management CLI Commands**

```
/core-service=management/security-realm=Kerberos:add
/core-service=management/security-realm=Kerberos/server-identity=kerberos:add
/core-service=management/security-realm=Kerberos/server-identity=kerberos/keytab=remote\/test-
server.elytron.org@ELYTRON.ORG:add(path=path/to/remote-test-server.keytab, debug=true)
/core-service=management/security-realm=Kerberos/authentication=kerberos:add(remove-
realm=true)
```

### Example: Kerberos Remoting Security Realm Configuration

```
<management>
  <security-realms>
    ...
    <security-realm name="Kerberos">
      <server-identities>
        <kerberos>
          <keytab principal="remote/test-server.elytron.org@ELYTRON.ORG" path="path/to/remote-test-
server.keytab" debug="true"/>
        </kerberos>
      </server-identities>
      <authentication>
        <kerberos remove-realm="true"/>
      </authentication>
    </security-realm>
  </security-realms>
  ...
</management>
```

#### Migrate the Kerberos Remoting SASL Authentication to Elytron

The steps to define the equivalent Elytron configuration are very similar to those described in Migrate Kerberos HTTP Authentication.

1. Define a security realm to be used to load identity information.

```
/path=kerberos:add(relative-to=user.home, path=src/kerberos)
/subsystem=elytron/properties-realm=kerberos-properties:add(users-properties=
{path=kerberos-users.properties, relative-to=kerberos, digest-realm-name=ELYTRON.ORG},
groups-properties={path=kerberos-groups.properties, relative-to=kerberos})
```

2. Define the Kerberos security factory for the server's identity.

```
/subsystem=elytron/kerberos-security-factory=test-server:add(relative-to=kerberos,
path=remote-test-server.keytab, principal=remote/test-server.elytron.org@ELYTRON.ORG)
```

3. Define the security domain and a SASL authentication factory.

```
/subsystem=elytron/security-domain=KerberosDomain:add(default-realm=kerberos-
properties, realms=[{realm=kerberos-properties, role-decoder=groups-to-roles}], permission-
mapper=default-permission-mapper)
/subsystem=elytron/sasl-authentication-factory=gssapi-authentication-factory:add(security-
domain=KerberosDomain, sasl-server-factory=elytron, mechanism-configurations=
[{mechanism-name=GSSAPI, credential-security-factory=test-server}])
```

This results in the following configuration in the **elytron** subsystem of the server configuration file.

```
<subsystem xmlns="urn:wildfly:elytron:4.0" final-providers="combined-providers" disallowed-
providers="OracleUcrypto">
  ...
  <security-domains>
    ...
    <security-domain name="KerberosDomain" default-realm="kerberos-properties" permission-
mapper="default-permission-mapper">
      <realm name="kerberos-properties" role-decoder="groups-to-roles"/>
    </security-domain>
  </security-domains>
  <security-realms>
   ...
     <properties-realm name="kerberos-properties">
       <users-properties path="kerberos-users.properties" relative-to="kerberos" digest-realm-
name="ELYTRON.ORG"/>
       <groups-properties path="kerberos-groups.properties" relative-to="kerberos"/>
     </properties-realm>
   </security-realms>
   <credential-security-factories>
     <kerberos-security-factory name="test-server" principal="remote/test-
server.elytron.org@ELYTRON.ORG" path="remote-test-server.keytab" relative-to="kerberos"/>
   </credential-security-factories>
   ...
   <sasl>
     ...
     <sasl-authentication-factory name="gssapi-authentication-factory" sasl-server-factory="elytron"
security-domain="KerberosDomain">
       <mechanism-configuration>
         <mechanism mechanism-name="GSSAPI" credential-security-factory="test-server"/>
       </mechanism-configuration>
     </sasl-authentication-factory>
     ...
   </sasl>
 </subsystem>
```

The management interface or remoting connectors can now be updated to reference the SASL authentication factory.

The two Elytron examples defined here could also be combined to use a shared security domain and security realm and just use protocol-specific authentication factories each referencing their own Kerberos security factory.

### 7.3.5. Migrate Composite Stores to Elytron

This section describes how to migrate a PicketBox or legacy security realm configuration that uses multiple identity stores to Elytron. When using either PicketBox or the legacy security realms, it is possible to define a configuration where authentication is performed against one identity store while the information used for authorization is loaded from a different store. When migrating to Elytron, this can be achieved by using an aggregate security realm.

The following examples perform user authentication using the **example-users.properties** properties file, and then query LDAP to load the group and role information.

NOTE

The configurations shown are based on the examples in the following sections, which provide additional background information:

- Migrate Properties-based Authentication and Authorization to Elytron
- Migrate LDAP Authentication Configuration to Elytron

**PicketBox Composite Store Configuration**
The PicketBox security domain for this scenario is configured using the following management CLI commands.

### Example: PicketBox Configuration Commands

```
/subsystem=security/security-domain=application-security:add

/subsystem=security/security-domain=application-security/authentication=classic:add(login-modules=
[ {code=UsersRoles, flag=Required, module-options={ password-stacking=useFirstPass,
usersProperties=file://${jboss.server.config.dir}/example-users.properties}} {code=LdapExtended,
flag=Required, module-options={ password-stacking=useFirstPass,
java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory,
java.naming.provider.url=ldap://localhost:10389, java.naming.security.authentication=simple,
bindDN="uid=admin,ou=system", bindCredential=secret, baseCtxDN="ou=users,dc=group-to-
principal,dc=wildfly,dc=org", baseFilter="(uid={0})", rolesCtxDN="ou=groups,dc=group-to-
principal,dc=wildfly,dc=org",roleFilter="(uniqueMember={1})", roleAttributeID="uid" }}])
```

This results in the following server configuration.

### Example: PicketBox Security Domain Configuration

```
<security-domain name="application-security">
  <authentication>
    <login-module code="UsersRoles" flag="required">
      <module-option name="password-stacking" value="useFirstPass"/>
      <module-option name="usersProperties" value="file://${jboss.server.config.dir}/example-
users.properties"/>
    </login-module>
    <login-module code="LdapExtended" flag="required">
      <module-option name="password-stacking" value="useFirstPass"/>
      <module-option name="java.naming.factory.initial" value="com.sun.jndi.ldap.LdapCtxFactory"/>
      <module-option name="java.naming.provider.url" value="ldap://localhost:10389"/>
      <module-option name="java.naming.security.authentication" value="simple"/>
      <module-option name="bindDN" value="uid=admin,ou=system"/>
      <module-option name="bindCredential" value="secret"/>
      <module-option name="baseCtxDN" value="ou=users,dc=group-to-principal,dc=wildfly,dc=org"/>
      <module-option name="baseFilter" value="(uid={0})"/>
      <module-option name="rolesCtxDN" value="ou=groups,dc=group-to-principal,dc=wildfly,dc=org"/>
      <module-option name="roleFilter" value="(uniqueMember={1})"/>
      <module-option name="roleAttributeID" value="uid"/>
    </login-module>
  </authentication>
</security-domain>
```

See Elytron Aggregate Security Realm Configuration for how to configure an aggregate security realm in the **elytron** subsystem to accomplish this.

**Legacy Security Realm Composite Store Configuration**

The legacy security realm configuration for this scenario is configured using the following management CLI commands.

### Example: Legacy Security Realm Configuration Commands

```
/core-service=management/ldap-connection=MyLdapConnection:add(url="ldap://localhost:10389",
search-dn="uid=admin,ou=system", search-credential="secret")

/core-service=management/security-realm=ApplicationSecurity:add
/core-service=management/security-
realm=ApplicationSecurity/authentication=properties:add(path=example-users.properties, relative-
to=jboss.server.config.dir, plain-text=true)

batch
/core-service=management/security-
realm=ApplicationSecurity/authorization=ldap:add(connection=MyLdapConnection)
/core-service=management/security-realm=ApplicationSecurity/authorization=ldap/username-to-
dn=username-filter:add(attribute=uid, base-dn="ou=users,dc=group-to-principal,dc=wildfly,dc=org")
/core-service=management/security-realm=ApplicationSecurity/authorization=ldap/group-
search=group-to-principal:add(base-dn="ou=groups,dc=group-to-principal,dc=wildfly,dc=org",
iterative=true, prefer-original-connection=true, principal-attribute=uniqueMember, search-
by=DISTINGUISHED_NAME, group-name=SIMPLE, group-name-attribute=uid)
run-batch
```

This results in the following server configuration.

### Example: Legacy Security Realm Configuration

```
<security-realms>
  ...
  <security-realm name="ApplicationSecurity">
    <authentication>
      <properties path="example-users.properties" relative-to="jboss.server.config.dir" plain-
text="true"/>
    </authentication>
    <authorization>
      <ldap connection="MyLdapConnection">
        <username-to-dn>
          <username-filter base-dn="ou=users,dc=group-to-principal,dc=wildfly,dc=org" attribute="uid"/>
        </username-to-dn>
        <group-search group-name="SIMPLE" iterative="true" group-name-attribute="uid">
          <group-to-principal search-by="DISTINGUISHED_NAME" base-dn="ou=groups,dc=group-to-
principal,dc=wildfly,dc=org" prefer-original-connection="true">
            <membership-filter principal-attribute="uniqueMember"/>
          </group-to-principal>
        </group-search>
      </ldap>
    </authorization>
  </security-realm>
</security-realms>
<outbound-connections>
```

```
  <ldap name="MyLdapConnection" url="ldap://localhost:10389" search-dn="uid=admin,ou=system"
search-credential="secret"/>
</outbound-connections>
```

See Elytron Aggregate Security Realm Configuration for how to configure an aggregate security realm in the **elytron** subsystem to accomplish this.

**Elytron Aggregate Security Realm Configuration**

The equivalent Elytron configuration for this scenario is configured using the following management CLI commands.

### Example: Elytron Configuration Commands

```
/subsystem=elytron/dir-context=ldap-connection:add(url=ldap://localhost:10389,
principal="uid=admin,ou=system", credential-reference={clear-text=secret})

/subsystem=elytron/ldap-realm=ldap-realm:add(dir-context=ldap-connection, direct-verification=true,
identity-mapping={search-base-dn="ou=users,dc=group-to-principal,dc=wildfly,dc=org", rdn-
identifier="uid", attribute-mapping=[{filter-base-dn="ou=groups,dc=group-to-
principal,dc=wildfly,dc=org",filter="(uniqueMember={1})",from="uid",to="Roles"}]})

/subsystem=elytron/properties-realm=application-properties:add(users-properties={path=example-
users.properties, relative-to=jboss.server.config.dir, plain-text=true, digest-realm-name="Application
Security"})

/subsystem=elytron/aggregate-realm=combined-realm:add(authentication-realm=application-
properties, authorization-realm=ldap-realm)

/subsystem=elytron/security-domain=application-security:add(realms=[{realm=combined-realm}],
default-realm=combined-realm, permission-mapper=default-permission-mapper)
/subsystem=elytron/http-authentication-factory=application-security-http:add(http-server-mechanism-
factory=global, security-domain=application-security, mechanism-configurations=[{mechanism-
name=BASIC}])
```

This results in the following server configuration.

### Example: Elytron Configuration

```
<subsystem xmlns="urn:wildfly:elytron:4.0" final-providers="combined-providers" disallowed-
providers="OracleUcrypto">
  ...
  <security-domains>
    ...
    <security-domain name="application-security" default-realm="combined-realm" permission-
mapper="default-permission-mapper">
      <realm name="combined-realm"/>
    </security-domain>
  </security-domains>
  <security-realms>
    <aggregate-realm name="combined-realm" authentication-realm="application-properties"
authorization-realm="ldap-realm"/>
      ...
      <properties-realm name="application-properties">
        <users-properties path="example-users.properties" relative-to="jboss.server.config.dir" digest-
realm-name="Application Security" plain-text="true"/>
```

```
      </properties-realm>
      <ldap-realm name="ldap-realm" dir-context="ldap-connection" direct-verification="true">
        <identity-mapping rdn-identifier="uid" search-base-dn="ou=users,dc=group-to-
principal,dc=wildfly,dc=org">
          <attribute-mapping>
            <attribute from="uid" to="Roles" filter="(uniqueMember={1})" filter-base-
dn="ou=groups,dc=group-to-principal,dc=wildfly,dc=org"/>
          </attribute-mapping>
        </identity-mapping>
      </ldap-realm>
  </security-realms>
  ...
  <http>
    ...
    <http-authentication-factory name="application-security-http" http-server-mechanism-
factory="global" security-domain="application-security">
      <mechanism-configuration>
        <mechanism mechanism-name="BASIC"/>
      </mechanism-configuration>
    </http-authentication-factory>
    ...
  </http>
  ...
  <dir-contexts>
    <dir-context name="ldap-connection" url="ldap://localhost:10389"
principal="uid=admin,ou=system">
      <credential-reference clear-text="secret"/>
    </dir-context>
  </dir-contexts>
</subsystem>
```

In the **elytron** subsystem, an **aggregate-realm** has been defined that specifies which security realms to use for authentication and which to use for authorization decisions.

### 7.3.6. Migrate Security Domains That Use Caching to Elytron

When using PicketBox, it is possible to define a security domain and enable in-memory caching for its access. This allows you to access the identity data in memory and avoids additional direct access to the identity store. It is possible to achieve a similar configuration with Elytron. This section describes how to configure security domain caching when using Elytron.

**PicketBox Cached Security Domain Configuration**
The following commands show how to configure a PicketBox security domain that enables caching.

#### Example: PicketBox Cached Security Domain Commands

```
/subsystem=security/security-domain=application-security:add(cache-type=default)
/subsystem=security/security-domain=application-security/authentication=classic:add(login-modules=
[{code=LdapExtended, flag=Required, module-options={
java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory,
java.naming.provider.url=ldap://localhost:10389, java.naming.security.authentication=simple,
bindDN="uid=admin,ou=system", bindCredential=secret, baseCtxDN="ou=users,dc=group-to-
principal,dc=wildfly,dc=org", baseFilter="(uid={0})", rolesCtxDN="ou=groups,dc=group-to-
principal,dc=wildfly,dc=org", roleFilter="(uniqueMember={1})", roleAttributeID="uid" }}])
```

This results in the following server configuration.

### Example: PicketBox Cached Security Domain Configuration

```
<subsystem xmlns="urn:jboss:domain:security:2.0">
  <security-domains>
    ...
    <security-domain name="application-security" cache-type="default">
      <authentication>
        <login-module code="LdapExtended" flag="required">
          <module-option name="java.naming.factory.initial" value="com.sun.jndi.ldap.LdapCtxFactory"/>
          <module-option name="java.naming.provider.url" value="ldap://localhost:10389"/>
          <module-option name="java.naming.security.authentication" value="simple"/>
          <module-option name="bindDN" value="uid=admin,ou=system"/>
          <module-option name="bindCredential" value="secret"/>
          <module-option name="baseCtxDN" value="ou=users,dc=group-to-
principal,dc=wildfly,dc=org"/>
          <module-option name="baseFilter" value="(uid={0})"/>
          <module-option name="rolesCtxDN" value="ou=groups,dc=group-to-
principal,dc=wildfly,dc=org"/>
          <module-option name="roleFilter" value="(uniqueMember={1})"/>
          <module-option name="roleAttributeID" value="uid"/>
        </login-module>
      </authentication>
    </security-domain>
  </security-domains>
</subsystem>
```

#### NOTE

This command and resulting configuration is similar to the example shown in Migrate LDAP Authentication Configuration to Elytron; however, here the attribute **cache-type** is defined with a value of **default**. The **default** cache type is an in-memory cache. When using PicketBox, you can also specify a **cache-type** of **infinispan**, however this type is not supported with Elytron.

### Elytron Cached Security Domain Configuration

Follow the steps below to create a similar configuration that caches a security domain when using Elytron.

1. Define a security realm and wrap the security realm in a caching realm. The caching realm can then be used in a security domain and subsequently in an authentication factory.

   #### Example: Elytron Security Realm Configuration Commands

   ```
   /subsystem=elytron/dir-context=ldap-connection:add(url=ldap://localhost:10389,
   principal="uid=admin,ou=system", credential-reference={clear-text=secret})
   /subsystem=elytron/ldap-realm=ldap-realm:add(dir-context=ldap-connection, direct-
   verification=true, identity-mapping={search-base-dn="ou=users,dc=group-to-
   principal,dc=wildfly,dc=org", rdn-identifier="uid", attribute-mapping=[{filter-base-
   dn="ou=groups,dc=group-to-principal,dc=wildfly,dc=org",filter="(uniqueMember=
   {1})",from="uid",to="Roles"}]})
   /subsystem=elytron/caching-realm=cached-ldap:add(realm=ldap-realm)
   ```

2. Define a security domain and an HTTP authentication factory that use the **cached-ldap** realm defined in the previous step.

   **Example: Elytron Security Domain and Authentication Factory Configuration Commands**

   ```
   /subsystem=elytron/security-domain=application-security:add(realms=[{realm=cached-ldap}],
   default-realm=cached-ldap, permission-mapper=default-permission-mapper)
   /subsystem=elytron/http-authentication-factory=application-security-http:add(http-server-
   mechanism-factory=global, security-domain=application-security, mechanism-
   configurations=[{mechanism-name=BASIC}])
   ```

   NOTE

   In this step, it is important that you reference the **caching-realm** instead of the original realm. Otherwise, caching is bypassed.

These commands result in the following additions to the server configuration.

**Example: Elytron Cached Security Domain Configuration**

```
<subsystem xmlns="urn:wildfly:elytron:4.0" final-providers="combined-providers" disallowed-
providers="OracleUcrypto">
  ...
  <security-domains>
    ...
    <security-domain name="application-security" default-realm="cached-ldap" permission-
mapper="default-permission-mapper">
      <realm name="cached-ldap"/>
    </security-domain>
  </security-domains>
  ...
  <security-realms>
    ....
  <ldap-realm name="ldap-realm" dir-context="ldap-connection" direct-verification="true">
      <identity-mapping rdn-identifier="uid" search-base-dn="ou=users,dc=group-to-
principal,dc=wildfly,dc=org">
        <attribute-mapping>
          <attribute from="uid" to="Roles" filter="(uniqueMember={1})" filter-base-
dn="ou=groups,dc=group-to-principal,dc=wildfly,dc=org"/>
        </attribute-mapping>
      </identity-mapping>
    </ldap-realm>
    <caching-realm name="cached-ldap" realm="ldap-realm"/>
  </security-realms>
  ...
  <http>
    ...
    <http-authentication-factory name="application-security-http" http-server-mechanism-
factory="global" security-domain="application-security">
      <mechanism-configuration>
        <mechanism mechanism-name="BASIC"/>
      </mechanism-configuration>
    </http-authentication-factory>
```

```
    ...
  </http>
   ...
  <dir-contexts>
    <dir-context name="ldap-connection" url="ldap://localhost:10389"
principal="uid=admin,ou=system">
      <credential-reference clear-text="secret"/>
    </dir-context>
  </dir-contexts>
  ...
```

### 7.3.7. Migrate JACC Security to Elytron

By default, JBoss EAP uses the legacy **security** subsystem to configure the Java Authorization Contract for Containers (JACC) policy provider and factory. The default configuration maps to implementations from PicketBox.

The **elytron** subsystem provides a built-in policy provider based on the JACC specification. Before you configure your server to allow Elytron to manage JACC configurations and other policies, you must first disable JACC in the legacy **security** subsystem by using the following management CLI command.

```
/subsystem=security:write-attribute(name=initialize-jacc, value=false)
```

Failure to do so can result in the following error in the server log: **MSC000004: Failure during stop of service org.wildfly.security.policy: java.lang.StackOverflowError**.

For information about how to enable JACC and define a JACC policy provider in the **elytron** subsystem, see Enabling JACC Using the **elytron** Subsystem in the *Development Guide* for JBoss EAP.

## 7.4. MIGRATE APPLICATION CLIENTS

### 7.4.1. Migrate a Naming Client Configuration to Elytron

This section describes how to migrate a client application that performs a remote JNDI lookup using an **org.jboss.naming.remote.client.InitialContext** class, which is backed by an **org.jboss.naming.remote.client.InitialContextFactory** class, to Elytron.

The following examples assume that the **InitialContextFactory** class is created by specifying properties for the user credentials and for the URL of the naming provider that it connects to.

**Example: InitialContext Code Used in the Previous Release**

```
Properties properties = new Properties();
properties.put(Context.INITIAL_CONTEXT_FACTORY,
"org.jboss.naming.remote.client.InitialContextFactory");
properties.put(Context.PROVIDER_URL,"http-remoting://127.0.0.1:8080");
properties.put(Context.SECURITY_PRINCIPAL, "bob");
properties.put(Context.SECURITY_CREDENTIALS, "secret");
InitialContext context = new InitialContext(properties);
Bar bar = (Bar) context.lookup("foo/bar");
...
```

You can choose from one of the following migration approaches:

- Migrate the Naming Client Using the Configuration File Approach
- Migrate the Naming Client Using the Programmatic Approach

#### 7.4.1.1. Migrate the Naming Client Using the Configuration File Approach

Follow these steps to migrate your naming client to Elytron using the configuration approach.

1. Create a **wildfly-config.xml** file in the client application **META-INF**/ directory. The file should contain the user credentials that are to be used when establishing a connection to the naming provider.

   **Example: wildfly-config.xml File**

   ```
   <configuration>
     <authentication-client xmlns="urn:elytron:client:1.2">
       <authentication-rules>
         <rule use-configuration="namingConfig">
           <match-host name="127.0.0.1"/>
         </rule>
       </authentication-rules>
       <authentication-configurations>
         <configuration name="namingConfig">
           <set-user-name name="bob"/>
           <credentials>
             <clear-password password="secret"/>
           </credentials>
         </configuration>
       </authentication-configurations>
     </authentication-client>
   </configuration>
   ```

2. Create an **InitialContext** as in the following example. Note that the **InitialContext** is backed by the **org.wildfly.naming.client.WildFlyInitialContextFactory** class.

   **Example: InitialContext Code**

   ```
   Properties properties = new Properties();
   properties.put(Context.INITIAL_CONTEXT_FACTORY,"org.wildfly.naming.client.WildFlyInitial
   ContextFactory");
   properties.put(Context.PROVIDER_URL,"remote+http://127.0.0.1:8080");
   InitialContext context = new InitialContext(properties);
   Bar bar = (Bar) context.lookup("foo/bar");
   ...
   ```

#### 7.4.1.2. Migrate the Naming Client Using the Programmatic Approach

Using this approach, you provide the user credentials that are used to establish a connection to the naming provider directly in the application code.

**Example: Code Using the Programmatic Approach**

```
// Create the authentication configuration
AuthenticationConfiguration namingConfig =
```

```
AuthenticationConfiguration.empty().useName("bob").usePassword("secret");

// Create the authentication context
AuthenticationContext context =
AuthenticationContext.empty().with(MatchRule.ALL.matchHost("127.0.0.1"), namingConfig);

// Create a callable that creates and uses an InitialContext
Callable<Void> callable = () -> {
    Properties properties = new Properties();

properties.put(Context.INITIAL_CONTEXT_FACTORY,"org.wildfly.naming.client.WildFlyInitialContextF
actory");
    properties.put(Context.PROVIDER_URL,"remote+http://127.0.0.1:8080");
    InitialContext context = new InitialContext(properties);
    Bar bar = (Bar) context.lookup("foo/bar");
    ...
    return null;
};

// Use the authentication context to run the callable
context.runCallable(callable);
```

### 7.4.2. Migrate an EJB Client to Elytron

This migration example assumes that the client application is configured to invoke an EJB deployed to a remote server using a **jboss-ejb-client.properties** file. This file, which is located in the client application **META-INF/** directory, contains the following information needed to connect to the remote server.

#### Example: **jboss-ejb-client.properties** File

```
remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABLED=false
remote.connections=default
remote.connection.default.host=127.0.0.1
remote.connection.default.port = 8080
remote.connection.default.username=bob
remote.connection.default.password=secret
```

The client looks up the EJB and calls one of its methods using code similar to the following example.

#### Example: Client Code That Calls a Remote EJB

```
// Create an InitialContext
Properties properties = new Properties();
properties.put(Context.URL_PKG_PREFIXES, "org.jboss.ejb.client.naming");
InitialContext context = new InitialContext(properties);

// Look up the EJB and invoke one of its methods
RemoteCalculator statelessRemoteCalculator = (RemoteCalculator) context.lookup(
    "ejb:/ejb-remote-server-side//CalculatorBean!" + RemoteCalculator.class.getName());
int sum = statelessRemoteCalculator.add(101, 202);
```

You can choose from one of the following migration approaches:

- Migrate the EJB Client Using the Configuration File Approach

- Migrate the EJB Client Using the Programmatic Approach

### 7.4.2.1. Migrate the EJB Client Using the Configuration File Approach

Follow these steps to migrate your naming client to Elytron using the configuration approach.

1. Configure a **wildfly-config.xml** file in the client application **META-INF/** directory. The file should contain the user credentials that are to be used when establishing a connection to the naming provider.

   **Example: wildfly-config.xml File**

   ```
   <configuration>
     <authentication-client xmlns="urn:elytron:client:1.2">
       <authentication-rules>
         <rule use-configuration="ejbConfig">
           <match-host name="127.0.0.1"/>
         </rule>
       </authentication-rules>
       <authentication-configurations>
         <configuration name="ejbConfig">
           <set-user-name name="bob"/>
           <credentials>
             <clear-password password="secret"/>
           </credentials>
         </configuration>
       </authentication-configurations>
     </authentication-client>
     <jboss-ejb-client xmlns="urn:jboss:wildfly-client-ejb:3.0">
       <connections>
         <connection uri="remote+http://127.0.0.1:8080" />
       </connections>
     </jboss-ejb-client>
   </configuration>
   ```

2. Create an **InitialContext** as in the following example. Note that the **InitialContext** is backed by the **org.wildfly.naming.client.WildFlyInitialContextFactory** class.

   **Example: InitialContext Code**

   ```
   // Create an InitialContext
   Properties properties = new Properties();
   properties.put(Context.INITIAL_CONTEXT_FACTORY,"org.wildfly.naming.client.WildFlyInitial
   ContextFactory");
   InitialContext context = new InitialContext(properties);

   // Look up an EJB and invoke one of its methods
   // Note that this code is the same as before
   RemoteCalculator statelessRemoteCalculator = (RemoteCalculator) context.lookup(
     "ejb:/ejb-remote-server-side//CalculatorBean!" + RemoteCalculator.class.getName());
   int sum = statelessRemoteCalculator.add(101, 202);----
   ```

3. You can now delete the obsolete **jboss-ejb-client.properties** file as that file is no longer needed.

#### 7.4.2.2. Migrate the EJB Client Using the Programmatic Approach

Using this approach, you provide the information needed to connect to the remote server directly in the application code.

**Example: Code Using the Programmatic Approach**

```
// Create the authentication configuration
AuthenticationConfiguration ejbConfig =
AuthenticationConfiguration.empty().useName("bob").usePassword("secret");

// Create the authentication context
AuthenticationContext context =
AuthenticationContext.empty().with(MatchRule.ALL.matchHost("127.0.0.1"), ejbConfig);

// Create a callable that invokes the EJB
Callable<Void> callable = () -> {

    // Create an InitialContext
    Properties properties = new Properties();
    properties.put(Context.INITIAL_CONTEXT_FACTORY,
"org.wildfly.naming.client.WildFlyInitialContextFactory");
    properties.put(Context.PROVIDER_URL, "remote+http://127.0.0.1:8080");
    InitialContext context = new InitialContext(properties);

    // Look up the EJB and invoke one of its methods
    // Note that this code is the same as before
    RemoteCalculator statelessRemoteCalculator = (RemoteCalculator) context.lookup(
        "ejb:/ejb-remote-server-side//CalculatorBean!" + RemoteCalculator.class.getName());
    int sum = statelessRemoteCalculator.add(101, 202);
    ...
    return null;
};

// Use the authentication context to run the callable
context.runCallable(callable);
```

You can now delete the obsolete **jboss-ejb-client.properties** file as that file is no longer needed.

## 7.5. MIGRATE SSL CONFIGURATIONS

### 7.5.1. Migrate a Simple SSL Configuration to Elytron

If you secured HTTP connections to the JBoss EAP server using a security realm, you can migrate that configuration to Elytron using the information provided in this section.

The following examples assume you have the following **keystore** configured in the **security-realm**.

**Example: SSL Configuration Using a Security Realm Keystore**

```
<security-realm name="ApplicationRealm">
  <server-identities>
    <ssl>
      <keystore path="server.keystore" relative-to="jboss.server.config.dir" keystore-
```

```
password="keystore_password" alias="server" key-password="key_password" />
    </ssl>
  </server-identities>
</security-realm>
```

Follow the steps below to achieve the same configuration using Elytron.

1. Create a **key-store** in the **elytron** subsystem that specifies the location of the keystore and the password by which it is encrypted. This command assumes the keystore was generated using the keytool command and its type is **JKS**.

   ```
   /subsystem=elytron/key-store=LocalhostKeyStore:add(path=server.keystore,relative-
   to=jboss.server.config.dir,credential-reference={clear-text="keystore_password"},type=JKS)
   ```

2. Create a **key-manager** in the **elytron** subsystem that specifies the **key-store** defined in the previous step, the alias, and password of the key.

   ```
   /subsystem=elytron/key-manager=LocalhostKeyManager:add(key-
   store=LocalhostKeyStore,alias-filter=server,credential-reference={clear-
   text="key_password"})
   ```

3. Create a **server-ssl-context** in the **elytron** subsystem that references the **key-manager** that was defined in the previous step.

   ```
   /subsystem=elytron/server-ssl-context=LocalhostSslContext:add(key-
   manager=LocalhostKeyManager)
   ```

4. Switch the **https-listener** from the legacy **security-realm** to the newly created Elytron **ssl-context**.

   ```
   batch
   /subsystem=undertow/server=default-server/https-listener=https:undefine-
   attribute(name=security-realm)
   /subsystem=undertow/server=default-server/https-listener=https:write-attribute(name=ssl-
   context,value=LocalhostSslContext)
   run-batch
   ```

5. Reload the server.

   ```
   reload
   ```

This results in the following **elytron** subsystem configuration in the server configuration file.

```
<subsystem xmlns="urn:wildfly:elytron:4.0" ...>
  ...
  <tls>
    <key-stores>
      <key-store name="LocalhostKeyStore">
        <credential-reference clear-text="keystore_password"/>
        <implementation type="JKS"/>
        <file path="server.keystore" relative-to="jboss.server.config.dir"/>
      </key-store>
    </key-stores>
    <key-managers>
```

```
      <key-manager name="LocalhostKeyManager" key-store="LocalhostKeyStore"  alias-
filter="server">
        <credential-reference clear-text="key_password"/>
      </key-manager>
    </key-managers>
    <server-ssl-contexts>
      <server-ssl-context name="LocalhostSslContext" key-manager="LocalhostKeyManager"/>
    </server-ssl-contexts>
  </tls>
</subsystem>
```

This results in the following **undertow** subsystem configuration in the server configuration file.

```
<https-listener name="https" socket-binding="https" ssl-context="LocalhostSslContext" enable-
http2="true"/>
```

For more information, see Elytron Subsystem and How to Secure the Management Interfaces in *How to Configure Server Security* for JBoss EAP.

### 7.5.2. Migrate CLIENT-CERT SSL Authentication to Elytron

To enable **CLIENT-CERT** SSL authentication, add a **truststore** element to the **authentication** element.

```
<security-realm name="ManagementRealm">
  <server-identities>
    <ssl>
      <keystore path="server.keystore" relative-to="jboss.server.config.dir" keystore-
password="KEYSTORE_PASSWORD" alias="server" key-password="key_password" />
    </ssl>
  </server-identities>
  <authentication>
    <truststore path="server.truststore" relative-to="jboss.server.config.dir" keystore-
password="TRUSTSTORE_PASSWORD" />
    <local default-user="$local"/>
    <properties path="mgmt-users.properties" relative-to="jboss.server.config.dir"/>
  </authentication>
</security-realm>
```

**NOTE**

With this configuration if the **CLIENT-CERT** authentication does not occur, clients can fall back to use either the local mechanism or the **username/password** authentication mechanism. To make **CLIENT-CERT** based authentication mandatory, remove the **local** and **properties** elements.

A legacy **truststore** can be used in two ways:

- Legacy **truststore** containing only CA
- Legacy **truststore** containing client's certificate

**Legacy truststore Containing Only CA**
Follow these steps to configure the server to prevent users without a valid certificate and private key from accessing the server using Elytron.

1. Create a **key-store** in the **elytron** subsystem that specifies the location of the keystore and the password by which it is encrypted. This command assumes the keystore was generated using the keytool command and its type is **JKS**.

   ```
   /subsystem=elytron/key-store=LocalhostKeyStore:add(path=server.keystore,relative-
   to=jboss.server.config.dir,credential-reference={clear-text="keystore_password"},type=JKS)
   ```

2. Create a **key-store** in the **elytron** subsystem that specifies the location of the truststore and the password by which it is encrypted. This command assumes the keystore was generated using the keytool command and its type is **JKS**.

   ```
   /subsystem=elytron/key-store=TrustStore:add(path=server.truststore,relative-
   to=jboss.server.config.dir,credential-reference={clear-text="truststore_password"},type=JKS)
   ```

3. Create a **key-manager** in the **elytron** subsystem that specifies the previously defined **LocalhostKeyStore** keystore, the alias, and password of the key.

   ```
   /subsystem=elytron/key-manager=LocalhostKeyManager:add(key-
   store=LocalhostKeyStore,alias-filter=server,credential-reference={clear-
   text="key_password"})
   ```

4. Create a **trust-manager** in the **elytron** subsystem that specifies the **key-store** of the previously created truststore.

   ```
   /subsystem=elytron/trust-manager=TrustManager:add(key-store=TrustStore)
   ```

5. Create a **server-ssl-context** in the **elytron** subsystem that references the previously defined **key-manager**, sets the **trust-manager** attribute, and enables client authentication.

   ```
   /subsystem=elytron/server-ssl-context=LocalhostSslContext:add(key-
   manager=LocalhostKeyManager,trust-manager=TrustManager,need-client-auth=true)
   ```

6. Switch the **https-listener** from the legacy **security-realm** to the newly created Elytron **ssl-context**.

   ```
   batch
   /subsystem=undertow/server=default-server/https-listener=https:undefine-
   attribute(name=security-realm)
   /subsystem=undertow/server=default-server/https-listener=https:write-attribute(name=ssl-
   context,value=LocalhostSslContext)
   run-batch
   ```

7. Reload the server.

   ```
   reload
   ```

This results in the following **elytron** subsystem configuration in the server configuration file.

```
<subsystem xmlns="urn:wildfly:elytron:4.0"...>
  ...
  <tls>
    <key-stores>
      <key-store name="LocalhostKeyStore">
```

```
      <credential-reference clear-text="keystore_password"/>
      <implementation type="JKS"/>
      <file path="server.keystore" relative-to="jboss.server.config.dir"/>
    </key-store>
    <key-store name="TrustStore">
      <credential-reference clear-text="truststore_password"/>
      <implementation type="JKS"/>
      <file path="server.truststore" relative-to="jboss.server.config.dir"/>
    </key-store>
  </key-stores>
  <key-managers>
    <key-manager name="LocalhostKeyManager" key-store="LocalhostKeyStore" alias-
filter="server">
      <credential-reference clear-text="key_password"/>
    </key-manager>
  </key-managers>
  <trust-managers>
    <trust-manager name="TrustManager" key-store="TrustStore"/>
  </trust-managers>
  <server-ssl-contexts>
    <server-ssl-context name="LocalhostSslContext" need-client-auth="true" key-
manager="LocalhostKeyManager" trust-manager="TrustManager"/>
  </server-ssl-contexts>
 </tls>
</subsystem>
```

This results in the following **undertow** subsystem configuration in the server configuration file.

```
<subsystem xmlns="urn:jboss:domain:undertow:7.0">
...
<https-listener name="https" socket-binding="https" ssl-context="LocalhostSslContext" enable-
http2="true"/>
...
</subsystem>
```

**Realms and Domains**

To allow using the predefined Elytron **ManagementDomain** security domain and **ManagementRealm** security realm, users are stored in standard properties files.

```
<security-domains>
  <security-domain name="ManagementDomain" default-realm="ManagementRealm" permission-
mapper="default-permission-mapper">
      <realm name="ManagementRealm" role-decoder="groups-to-roles"/>
      <realm name="local"/>
  </security-domain>
</security-domains>
<security-realms>
  <properties-realm name="ManagementRealm">
      <users-properties path="mgmt-users.properties" relative-to="jboss.server.config.dir" digest-
realm-name="ManagementRealm"/>
      <groups-properties path="mgmt-groups.properties" relative-to="jboss.server.config.dir"/>
  </properties-realm>
</security-realms>
```

The security realm is used in two situations:

- When certificate authentication fails, the security realm is used in password fallback case.
- When authorization is done for password as well as certificate, the realm provides the roles of individual users.

Thus, for any client certificate, a user must exist in the security realm.

**Principal Decoder**

When certificate authentication is used and the security realm accepts user names to resolve an identity, there has to be a defined way to obtain the **username** from a client certificate.

In this case the **CN** attribute is used in the certificate subject.

```
/subsystem=elytron/x500-attribute-principal-decoder=x500-decoder:add(attribute-name=CN)
```

**HTTP Authentication Factory**

For the HTTP connections, an HTTP authentication factory is defined, using the previously defined resources. It is configured to support **CLIENT_CERT** and **DIGEST** authentication.

Since a properties realm only verifies passwords and is not able to verify client certificates, you need to first add a configuring mechanism factory. This disables certificate verification against the security realm.

```
/subsystem=elytron/configurable-http-server-mechanism-factory=configured-cert:add(http-server-
mechanism-factory=global, properties={org.wildfly.security.http.skip-certificate-verification=true})
```

The HTTP authentication can be created as:

```
./subsystem=elytron/http-authentication-factory=client-cert-digest:add(http-server-mechanism-
factory=configured-cert,security-domain=ManagementDomain,mechanism-configurations=
[{mechanism-name=CLIENT_CERT,pre-realm-principal-transformer=x500-decoder},{mechanism-
name=DIGEST, mechanism-realm-configurations=[{realm-name=ManagementRealm}]}])
```

The above command results in:

```
<subsystem xmlns="urn:wildfly:elytron:4.0" final-providers="combined-providers" disallowed-
providers="OracleUcrypto">
  ...
  <http>
    ...
    <http-authentication-factory name="client-cert-digest" http-server-mechanism-factory="configured-
cert" security-domain="ManagementDomain">
      <mechanism-configuration>
        <mechanism mechanism-name="CLIENT_CERT" pre-realm-principal-transformer="x500-
decoder"/>
        <mechanism mechanism-name="DIGEST">
          <mechanism-realm realm-name="ManagementRealm"/>
        </mechanism>
      </mechanism-configuration>
    </http-authentication-factory>
    ...
    <configurable-http-server-mechanism-factory name="configured-cert" http-server-mechanism-
factory="configured-cert">
        <properties>
            <property name="org.wildfly.security.http.skip-certificate-verification" value="true"/>
        </properties>
    </configurable-http-server-mechanism-factory>
    ...
  </http>
  ...
</subsystem>
```

# CHAPTER 8. MIGRATING FROM OLDER RELEASES OF JBOSS EAP

## 8.1. MIGRATING FROM JBOSS EAP 5 TO JBOSS EAP 7

This guide focuses on the changes that are required to successfully run and deploy JBoss EAP 6 applications on JBoss EAP 7. If you plan to migrate your applications directly from JBoss EAP 5 to JBoss EAP 7, there are a number of resources available to help you plan and execute your migration. We suggest you take the following approach.

1. See Summary of Changes Made to Each Release in this guide for a quick, high-level overview of the changes made to each release of JBoss EAP.

2. Read through the JBoss EAP 6 *Migration Guide* and this guide to become familiar with the contents of each one.

3. Use the JBoss EAP 5 Component Upgrade Reference as a quick reference to migration information about specific components and features.

4. The rule-based Red Hat Application Migration Toolkit continues to add rules to help you migrate directly from JBoss EAP 5 to JBoss EAP 7. You can use these tools to analyze your application and to generate detailed reports about the changes needed to migrate to JBoss EAP 7. For more information, see Use Red Hat Application Migration Toolkit to Analyze Applications for Migration.

5. The Customer Portal Knowledgebase currently contains articles and solutions to help with migration from JBoss EAP 5 to JBoss EAP 6. There are plans in place to add additional content for migration from JBoss EAP 5 to JBoss EAP 7 over time.

## 8.2. SUMMARY OF CHANGES MADE TO EACH RELEASE

Before you plan your migration, you should be aware of the changes that were made to JBoss EAP 6 and JBoss EAP 7.

The JBoss EAP 6 Migration Guide covers changes that were made between JBoss EAP 5 and JBoss EAP 6. The following is a condensed list of the most significant changes made in JBoss EAP 6.

- Implemented a new architecture built on the Modular Service Container
- Was a certified implementation of the Java Enterprise Edition 6 specification
- Introduced domain management, new deployment configuration, and a new file directory structure and scripts
- Standardized on new portable JNDI namespaces

See Review What's New and Different in JBoss EAP 6 in the JBoss EAP 6 *Migration Guide* for a detailed list of changes made in that release.

JBoss EAP 7 is built on the same modular structure as JBoss EAP 6 and includes the same domain management, deployment configuration, file directory structure, and scripts. It also still uses the same standardized JNDI namespaces. However, JBoss EAP 7 introduces the following changes.

- Adds support for the Java Enterprise Edition 7 specification

- Replaces the web server with Undertow
- Replaces the JacORB IIOP implementation with a downstream branch of the OpenJDK ORB
- Includes Apache ActiveMQ Artemis as the new messaging provider
- Removes the **cmp**, **jaxr**, and **threads** subsystems
- Removes support for EJB entity beans

For a more complete list of changes, see Review What's New in JBoss EAP 7

## 8.3. REVIEW THE CONTENT IN THE MIGRATION GUIDES

Review the entire contents of the Migration Guide for each release to become aware of the features that were added or deprecated, and to understand the server configuration and the application changes required to run existing applications for that release.

Because the underlying architecture was not changed between JBoss EAP 6 and JBoss EAP 7, many of the changes documented in the JBoss EAP 6 *Migration Guide* still apply. For example, changes documented under Changes Required by Most Applications are related to the underlying architectural changes made in JBoss EAP 6, which still apply to this release. The change to the new modular class loading system is significant and impacts the packaging and dependencies of almost every JBoss EAP 5 application. Many of the changes listed under Changes Dependent on Your Application Architecture and Components are also still valid. However, because JBoss EAP 7 replaced the web server, ORB, and messaging provider, removed the **cmp**, **threads**, and **jaxr** subsystems, and removed support for EJB entity beans, you must consult this guide for any changes related to those component areas. Pay particular attention to the Server Configuration Changes and Application Migration Changes detailed in this guide before you begin.

## 8.4. JBOSS EAP 5 COMPONENT UPGRADE REFERENCE

Use the following table to find information about how to migrate a particular feature or component from JBoss EAP 5 to JBoss EAP 7.2.

| JBoss EAP 5 Feature or Component | Summary of Changes and Where to Find Migration Information |
|---|---|
| Application Packaging and Class Loading | In JBoss EAP 6, the previous hierarchical class loading structure was replaced with a modular architecture based on JBoss Modules. Application packaging also changed due to the new modular class loading structure. This architecture is still used in JBoss EAP 7. For information about the new modular architecture, see the following chapter in the JBoss EAP 7.2 *Development Guide*.<br><br>• Class Loading and Modules<br><br>For information about how to update and repackage applications for the new modular architecture, see the following section in the JBoss EAP 6 *Migration Guide*.<br><br>• Class Loading Changes |

| JBoss EAP 5 Feature or Component | Summary of Changes and Where to Find Migration Information |
|---|---|
| Application Configuration Files | Due to the changes in JBoss EAP 6 to use modular class loading, you might need to create or modify one or more application configuration files to add dependencies or to prevent automatic dependencies from loading. This has not changed in JBoss EAP 7. For details, see the following section in the JBoss EAP 6 *Migration Guide*.<br><br>● Configuration File Changes |
| Caching and Infinispan | JBoss Cache was replaced by Infinispan for internal use by the server only in JBoss EAP 6. See the following sections in the JBoss EAP 6 *Migration Guide* for information about how to replace JBoss Cache in application code.<br><br>● Cache Changes<br><br>Infinispan caching strategy and configuration changes for JBoss EAP 7 are documented in the following section of this guide.<br><br>● Infinispan Server Configuration Changes |
| Data Sources and Resource Adapters | JBoss EAP 6 consolidated configuration of data sources and resource adapters into mainly one file and this is still true in JBoss EAP 7. See the following section in the JBoss EAP 6 *Migration Guide* for more information.<br><br>● Datasource and Resource Adapter Configuration Changes |
| Directory Structure, Scripts, and Deployment Configuration | In JBoss EAP 6, the directory structure, scripts, and deployment configuration changed. These changes are still valid in JBoss EAP 7. See the following section of the JBoss EAP 6 *Migration Guide* for more information.<br><br>● Review What's New and Different in JBoss EAP 6 |

| JBoss EAP 5 Feature or Component | Summary of Changes and Where to Find Migration Information |
|---|---|
| EJB | The Java EE 7 specification made EJB 2.x and earlier features optional, so it is strongly recommended that you rewrite your application code to use the EJB 3.x specification and JPA. For information about deprecated features and changes required to run EJB 2.x, see the following section in the JBoss EAP 6 *Migration Guide*.<br><br>● EJB 2.x and Earlier Changes<br><br>In JBoss EAP 6, stateful EJB cache and stateless session bean pool size is configured in the **ejb3** subsystem of the server configuration file. The**jboss-ejb3.xml** deployment descriptor replaces the **jboss.xml** deployment descriptor file. For more information about these changes, see the following section in the JBoss EAP 6 *Migration Guide*.<br><br>● EJB Changes<br><br>The default remote connector and port has changed in JBoss EAP 7. For more information about this and server configuration changes, see the following sections in this guide.<br><br>● EJB Server Configuration Changes<br>● Migrate EJB Client Code<br><br>EJB entity beans are not supported in JBoss EAP 7. For information about how to migrate entity beans to JPA, see the following section in this guide.<br><br>● Migrate Entity Beans to JPA |

| JBoss EAP 5 Feature or Component | Summary of Changes and Where to Find Migration Information |
|---|---|
| Hibernate and JPA | In JBoss EAP 6, Hibernate was updated from version 3 to version 4. This version of JBoss EAP also implemented the JPA 2.0 specification and changes were made to JPA persistence properties. For information about how to modify your application for these changes, see the following section in the JBoss EAP 6 *Migration Guide*.<br>• Hibernate and JPA Changes<br>JBoss EAP 7.2 implements JPA 2.2 and includes Hibernate 5.3. It also includes Hibernate Search version 5.10. Other changes include removal of support for EJB entity beans and additional updates to JPA persistence properties. For information about how these changes impact your applications, see the following sections in this guide.<br>• Hibernate and JPA Migration Changes<br>• Hibernate Search Changes<br>• Migrate Entity Beans to JPA<br>• JPA Persistence Property Changes<br>**NOTE**<br>Use of a different version of Hibernate than the one shipped with JBoss EAP is unsupported. The version shipped with JBoss EAP is the only version of Hibernate that is tested, and is the only version for which patches will be provided for defects. |
| JAX-RS and RESTEasy | JBoss EAP 6 bundled RESTEasy 2, which automatically configured RESTEasy and required changes in application configuration. See the following section in the JBoss EAP 6 *Migration Guide* for information.<br>• JAX-RS and RESTEasy Changes<br>JBoss EAP 7 includes RESTEasy 3 and many classes have been deprecated. The version of Jackson changed from version 1.9.9 to version 2.6.3 or greater. For details about these changes, see the following section in this guide.<br>• JAX-RS and RESTEasy Application Changes |
| JBoss AOP | JBoss AOP (Aspect Oriented Programming) was removed in JBoss EAP 6. For information about how to refactor applications that use JBoss AOP, see the following section in the JBoss EAP 6 *Migration Guide*.<br>• JBoss AOP Changes |

| JBoss EAP 5 Feature or Component | Summary of Changes and Where to Find Migration Information |
|---|---|
| JGroups and Clustering | The way you enable clustering and specify bind addresses changed in JBoss EAP 6. See the following section in the JBoss EAP 6 *Migration Guide* for more information. <br><br> ● Clustering Changes <br><br> In JBoss EAP 7, JGroups now defaults to using a private network interface instead of a public network interface and also introduces **<channel>** elements to the **jgroups** subsystem. JBoss EAP 7 also includes the Undertow mod_cluster implementation, introduces a new API for building singleton services, and other new clustering features. These changes are documented in the following sections of this guide. <br><br> ● JGroups Server Configuration Changes <br> ● Application Clustering Changes |
| JNDI | JBoss EAP 6 implemented a new standardized global JNDI namespace and a series of related namespaces that map to the various scopes of a Java EE application. See the following section of the JBoss EAP 6 *Migration Guide* for information about application changes needed to use the new JNDI namespace rules. <br><br> ● JNDI Changes |
| JSF | JBoss EAP 6.4 included both JSF 1.2 and JSF 2.1 and allowed you to configure your application to use the older version. This is no longer possible in JBoss EAP 7.2, which now includes JSF 2.3. See the following section in this guide for more information. <br><br> ● JavaServer Faces (JSF) Code Changes |
| Logging | JBoss EAP 6 introduced a new JBoss Logging framework that is still used in JBoss EAP 7. Applications that use third-party logging frameworks might be impacted by the modular class loading changes. Review the following section in the JBoss EAP 6 *Migration Guide* for information about these changes. <br><br> ● Logging Changes <br><br> In JBoss EAP 7, annotations in the **org.jboss.logging** package are now deprecated, which impacts source code and Maven GAVs (groupId:artifactId:version). The prefixes for all log messages were also changed. For more information about these changes, see the following sections in this guide. <br><br> ● JBoss Logging Changes <br> ● Logging Message Prefix Changes |

| JBoss EAP 5<br>Feature or Component | Summary of Changes and<br>Where to Find Migration Information |
|---|---|
| Messaging and JMS | In JBoss EAP 6, HornetQ replaced JBoss Messaging as the default JMS implementation. Then in JBoss EAP 7, ActiveMQ Artemis replaced HornetQ as the built-in messaging provider.<br><br>The best approach to migrating your messaging configuration is to start with the JBoss EAP 7 default server configuration and use the following guide to apply your current messaging configuration changes.<br><br>• *Configuring Messaging* for JBoss EAP 7.2<br><br>If you want to understand the changes required to move from JBoss Messaging to HornetQ, review the following section of the JBoss EAP 6 *Migration Guide*.<br><br>• HornetQ Changes<br><br>Then review the following information about how to migrate the HornetQ configuration and related messaging data in this guide.<br><br>• Messaging Server Configuration Changes<br>• Messaging Application Changes |
| ORB | In JBoss EAP 6, JacORB configuration was moved from the ***EAP_HOME*/server/production/conf/jacorb.properties** file to the server configuration file. JBoss EAP 7 then replaced the JacORB IIOP implementation with a downstream branch of the OpenJDK ORB.<br><br>The best approach to migrating your ORB configuration is to start with the JBoss EAP 7 default server configuration and use the following section in the JBoss EAP 7.2 *Configuration Guide* to apply the your current ORB configuration changes.<br><br>• ORB Configuration |
| Remote Invocation | A new EJB client API was introduced in JBoss EAP 6 for remote invocations; however, if you preferred not to rewrite your application code to use the new API, you could modify your existing code to use the **ejb:BEAN_REFERENCE** for remote access to EJBs. See the following section in the JBoss EAP 6 *Migration Guide* for more information.<br><br>• Remote Invocation Changes<br><br>In JBoss EAP 7, the default connector and default remote connection port changed. For more information, see the following sections in this guide.<br><br>• Update the Remote URL Connector and Port<br>• Update External Clients<br>• Migrate EJB Client Code |

| JBoss EAP 5 Feature or Component | Summary of Changes and Where to Find Migration Information |
|---|---|
| Seam 2.x | While official support for Seam 2.2 applications was dropped in JBoss EAP 6, it was still possible to configure dependencies for JSF 1.2 and Hibernate 3 to allow Seam 2.2 applications to run on that release. JBoss EAP 7.2, which now includes JSF 2.3 and Hibernate 5.3.1, does not support Seam 2.2 or Seam 2.3 due to end of life of Red Hat JBoss Web Framework Kit. It is recommended that you rewrite your Seam components using Weld CDI beans. |
| Security | Security updates in JBoss EAP 6 included changes to security domain names and changes to how to configure security for basic authentication. The LDAP security realm configuration was moved to the server configuration file. See the following sections in the JBoss EAP 6 *Migration Guide* for more information.<br>• Security Changes<br>• LDAP Security Realm Changes<br>Updates that impact security in JBoss EAP 7 include server configuration changes and application changes. Information can be found in the following sections of this guide.<br>• Security Server Configuration Changes<br>• Security Application Changes |
| Spring Applications | Spring 4.2.x is the earliest stable Spring version supported by JBoss EAP 7. For information about Apache CXF Spring web services and Spring RESTEasy integration changes, see the following sections in this guide.<br>• Apache CXF Spring Web Services Changes<br>• Spring RESTEasy Integration Changes |
| Transactions | JBoss EAP 6 consolidated transaction configuration and moved it to the server configuration file. Other updates included changes to JTA node identifier settings and how to enable JTS. For details, see the following section in the JBoss EAP 6 *Migration Guide*.<br>• JTS and JTA Changes<br>Some Transaction Manager configuration attributes that were available in the **transactions** subsystem in JBoss EAP 6 have changed in JBoss EAP 7. For more information, see the following section in this guide.<br>• Transactions Subsystem Changes |

| JBoss EAP 5 Feature or Component | Summary of Changes and Where to Find Migration Information |
|---|---|
| Valves | Undertow replaced JBoss Web in JBoss EAP 7 and valves are no longer supported. See the following sections in this guide.<br>• Migrate Global Valves<br>• Migrate Custom Application Valves<br>• Migrate Authenticator Valves |
| Web Services | JBoss EAP 6 included JBossWS 4. For information about the changes required by that version update, see the following section in the JBoss EAP 6 *Migration Guide*.<br>• Web Services Changes<br>JBoss EAP 7 introduced JBossWS 5. See the following section in this guide for required updates.<br>• Web Services Applications Changes |

# APPENDIX A. REFERENCE MATERIAL

## A.1. JACORB SUBSYSTEM MIGRATION OPERATION WARNINGS

The **migrate** operation is not able to process all resources and attributes. The following table lists some of the warnings you might see when you run either the **migrate** or **describe-migration** operation for the **jacorb** subsystem.

### NOTE

If you see "Could not migrate" or "Can not migrate" entries in the output of the **migrate** operation, this indicates the migration of the server configuration completed successfully but it was not able to automatically migrate all of the elements and attributes. You must follow the suggestions provided by the "migration-warnings" to modify those configurations.

| Warning Message | What It Means / How to Fix It |
|---|---|
| The **iiop** migration can be performed when the server is in **admin-only** mode | The **migrate** operation requires starting the server in **admin-only** mode, which is done by adding**--start-mode=admin-only** to the server start command:<br><br>`$ EAP_HOME/bin/standalone.sh --start-mode=admin-only` |
| Properties *X* cannot be emulated using OpenJDK ORB and are not supported | Configuration of the specified property is not supported and is not included in the new **iiop-openjdk** subsystem configuration. The behavior exhibited by this property in the previous release of JBoss EAP is not migrated and the administrator must verify that the new **iiop-openjdk** subsystem in JBoss EAP 7 is able to operate correctly without that behavior.<br><br>Unsupported properties include: **cache-poa-names**, **cache-typecodes**, **chunk-custom-rmi-valuetypes**, **client-timeout**, **comet**, **indirection-encoding-disable**, **iona**, **lax-boolean-encoding**, **max-managed-buf-size**, **max-server-connections**, **max-threads**, **outbuf-cache-timeout**, **outbuf-size**, **queue-max**, **queue-min**, **poa-monitoring**, **print-version**, **retries**, **retry-interval**, **queue-wait**, **server-timeout**, **strict-check-on-tc-creation**, **use-bom**, **use-imr**. |

| Warning Message | What It Means / How to Fix It |
|---|---|
| The properties *X* use expressions. Configuration properties that are used to resolve those expressions should be transformed manually to the new **iiop-openjdk** subsystem format | Properties that use expressions must be configured manually by the administrator.<br><br>For example, the **jacorb** subsystem in JBoss EAP 6 defined a **giop-minor-version** property. The **iiop-openjdk** subsystem in JBoss EAP 7 defines a **giop-version** property. Suppose the **jacorb** subsystem minor version attribute is set to **${iiop-giop-minor-version}** and the system property is configured in the **standalone.conf** file as **-Diiop-giop-minor-version=1**. After the **migrate** operation, the adminstrator must change the system property value to **1.1** to ensure the new subsystem is configured correctly. |
| Can not migrate: the new **iiop-openjdk** subsystem is already defined | The message contains the explanation. |

## A.2. MESSAGING SUBSYSTEM MIGRATION OPERATION WARNINGS

The **migrate** operation is not able to process all resources and attributes. The following table lists some of the warnings you might see when you run either the **migrate** or **describe-migration** operation for the **messaging** subsystem.

NOTE

If you see "Could not migrate" or "Can not migrate" entries in the output of the **migrate** operation, this indicates the migration of the server configuration completed successfully but it was not able to automatically migrate all of the elements and attributes. You must follow the suggestions provided by the "migration-warnings" to modify those configurations.

| Warning Message | What It Means / How to Fix It |
|---|---|
| The **migrate** operation can not be performed: the server must be in **admin-only** mode | The **migrate** operation requires starting the server in **admin-only** mode, which is done by adding **--start-mode=admin-only** to the server start command:<br><br>`$ EAP_HOME/bin/standalone.sh --start-mode=admin-only` |
| Can not migrate attribute **local-bind-address** from resource *X*. Use instead the **socket-binding** attribute to configure this **broadcast-group**. | The message contains the explanation and how to fix it. |

| Warning Message | What It Means / How to Fix It |
|---|---|
| Can not migrate attribute **local-bind-port** from resource *X*. Use instead the **socket-binding** attribute to configure this **broadcast-group**. | The message contains the explanation and how to fix it. |
| Can not migrate attribute **group-address** from resource *X*. Use instead the **socket-binding** attribute to configure this **broadcast-group**. | The message contains the explanation and how to fix it. |
| Can not migrate attribute **group-port** from resource *X*. Use instead the **socket-binding** attribute to configure this **broadcast-group**. | The **broadcast-group** resource no longer accepts the **local-bind-address**, **local-bind-port**, **group-address**, or **group-port** attributes. It only accepts a **socket-binding** attribute. The warning is notification that resource *X* has an unsupported attribute. You must manually set the **socket-binding** attribute on the resource and ensure it corresponds to a defined **socket-binding** resource. |
| Classes providing the *X* are discarded during the migration. To use them in the new **messaging-activemq** subsystem, you will have to extend the Artemis-based **Interceptor**. | Messaging interceptors support is significantly different in JBoss EAP 7. Any interceptors configured in the previous version of the subsystem are discarded during migration. See Migrate Messaging Interceptors for more information. |
| Can not migrate the HA configuration of *X*. Its **shared-store** and **backup** attributes holds expressions and it is not possible to determine unambiguously how to create the corresponding **ha-policy** for the messaging-activemq's server. | This means the **hornetq-server** *X*'s **shared-store** or **backup** attributes contained an expression, such as ${xxx}, and the migration operation was not able to resolve it to a concrete expression. The value is discarded and the **ha-policy** for the **messaging-activemq** must be updated manually. |
| Can not migrate attribute **local-bind-address** from resource *X*. Use instead the **socket-binding** attribute to configure this **discovery-group**. | The message contains the explanation and how to fix it. |
| Can not migrate attribute **local-bind-port** from resource *X*. Use instead the **socket-binding** attribute to configure this **discovery-group**. | The message contains the explanation and how to fix it. |
| Can not migrate attribute **group-address** from resource *X*. Use instead the **socket-binding** attribute to configure this **discovery-group**. | The message contains the explanation and how to fix it. |

| Warning Message | What It Means / How to Fix It |
|---|---|
| Can not migrate attribute **group-port** from resource *X*. Use instead the **socket-binding** attribute to configure this **discovery-group**. | The **discovery-group** resources no longer accept **local-bind-address**, **local-bind-port**, **group-address**, or **group-port** attributes. It only accepts a **socket-binding**. The warning is notification that resource *X* has an unsupported attribute. You must manually set the **socket-binding** attribute on the resource and ensures it corresponds to a defined **socket-binding** resource. |
| Can not create a **legacy-connection-factory** based on **connection-factory** *X*. It uses a HornetQ **in-vm** connector that is not compatible with Artemis **in-vm** connector | The legacy HornetQ remote **connection-factory** resources are migrated into **legacy-connection-factory** resources to allow JBoss EAP 6 clients to connect to JBoss EAP 7. However, **legacy-connection-factory** resources are only created when the **connection-factory** is using remote connectors. Any **connection-factory** using **in-vm** is not migrated because **in-vm** clients are based on JBoss EAP 7, not JBoss EAP 6. This warning is notification that the **in-vm connection-factory** was not migrated. |
| Can not migrate attribute *X* from resource *Y*. The attribute uses an expression that can be resolved differently depending on system properties. After migration, this attribute must be added back with an actual value instead of the expression. | This warning appears when the migration can not resolve attribute *X* to a concrete value during the migration process. The value is discarded and the attribute must be migrated manually. This happens in the following cases:<br><br>● **cluster-connection forward-when-no-consumers**:<br>This boolean attribute has been replaced by the **message-load-balancing-type** attribute, which is an enum with a value of **OFF**, **STRICT**, or **ON_DEMAND**.<br><br>● **broadcast-group** and **discovery-group**'s **jgroups-stack** and **jgroups-channel** attributes<br>They reference other resources and JBoss EAP 7 no longer accepts these expressions. |
| Can not migrate attribute *X* from resource *Y*. This attribute is not supported by the new **messaging-activemq** subsystem. | Some attributes are no longer supported in the new **messaging-activemq** subsystem and are simply discarded:<br><br>● **hornetq-server**'s **failback-delay**<br>● **http-connector**'s **use-nio** attribute<br>● **http-acceptor**'s **use-nio** attribute<br>● **remote-connector**'s **use-nio** attribute<br>● **remote-acceptor**'s **use-nio** attribute |

| Warning Message | What It Means / How to Fix It |
|---|---|
| Can not migrate attribute **failback-delay** from resource *X*. Artemis detects failback deterministically and it no longer requires to specify a delay for failback to occur. | The message contains the explanation. |

### Replace the Deprecated broadcast-group or discovery-group Attributes

If you are advised to replace the deprecated **broadcast-group** or **discovery-group** attributes with the **socket-binding** attribute, you can add the new attribute using the management CLI.

This example assumes you are migrating a standalone server that contains the following **discovery-group** configuration in the **messaging** subsystem.

```
<discovery-groups>
   <discovery-group name="my-discovery-group">
      <group-address>224.0.1.105</group-address>
      <group-port>56789</group-port>
   </discovery-group>
</discovery-groups>
```

When you run the **migrate** operation for the **messaging** subsystem, you see the following output and warnings:

```
/subsystem=messaging:migrate
{
   "outcome" => "success",
   "result" => {"migration-warnings" => [
      "WFLYMSG0084: Can not migrate attribute group-address from resource [
   (\"subsystem\" => \"messaging-activemq\"),
   (\"server\" => \"default\"),
   (\"discovery-group\" => \"my-discovery-group\")
]. Use instead the socket-binding attribute to configure this discovery-group.",
      "WFLYMSG0084: Can not migrate attribute group-port from resource [
   (\"subsystem\" => \"messaging-activemq\"),
   (\"server\" => \"default\"),
   (\"discovery-group\" => \"my-discovery-group\")
]. Use instead the socket-binding attribute to configure this discovery-group."
   ]}
}
```

The **migrate** operation creates a **discovery-group** named "my-discovery-group" in the new **messaging-activemq** subsystem that is now configured like the following.

```
<discovery-group name="my-discovery-group"/>
```

You must now use the following management CLI command to create a **socket-binding** element in the server configuration file named "my-discovery-group-socket-binding".

```
/socket-binding-group=standard-sockets/socket-binding=my-discovery-group-socket-
binding:add(multicast-address=224.0.1.105, multicast-port=56789)
```

Next, add the newly created **socket-binding** to the **discovery-group** named "my-discovery-group" in the **messaging-activemq** subsystem in the server configuration file using the following management CLI command.

```
/subsystem=messaging-activemq/server=default/discovery-group=my-discovery-group:write-
attribute(name=socket-binding,value=my-discovery-group-socket-binding)
```

These commands create the following XML in the server configuration file.

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:4.0">
    <server name="default">
        ...
        <discovery-group name="my-discovery-group" socket-binding="my-discovery-group-socket-
binding"/>
        ...
    </server>
</subsystem>
...
<socket-binding-group name="standard-sockets" default-interface="public" port-
offset="${jboss.socket.binding.port-offset:0}">
    ...
    <socket-binding name="my-discovery-group-socket-binding" multicast-address="224.0.1.105"
multicast-port="56789"/>
    ...
</socket-binding-group>
```

## A.3. WEB SUBSYSTEM MIGRATION OPERATION WARNINGS

The **migrate** operation is not able to process all resources and attributes. The following table lists some of the warnings you might see when you run either the **migrate** or **describe-migration** operation for the **web** subsystem.

**NOTE**

If you see "Could not migrate" or "Can not migrate" entries in the output of the **migrate** operation, this indicates the migration of the server configuration completed successfully but it was not able to automatically migrate all of the elements and attributes. You must follow the suggestions provided by the "migration-warnings" to modify those configurations.

| Warning Message | What It Means / How to Fix It |
|---|---|
| Migrate operation only allowed in admin only mode | The **migrate** operation requires starting the server in **admin-only** mode, which is done by adding parameter **--admin-only** to the server start command: `$ EAP_HOME/bin/standalone.sh --admin-only` |

| Warning Message | What It Means / How to Fix It |
|---|---|
| Could not migrate resource *X* | The behavior exhibited by this resource in the previous release of JBoss EAP was not migrated. The administrator must verify if the new **undertow** subsystem in JBoss EAP 7 is able to operate correctly without that behavior or whether the behavior must be migrated manually. |
| Could not migrate attribute *X* from resource *Y*. | The behavior exhibited by this resource attribute in the previous release of JBoss EAP was not migrated. The administrator must verify if the new **undertow** subsystem in JBoss EAP 7 is able to operate correctly without that behavior or whether the behavior must be migrated manually.<br><br>See Web Subsystem Migration Operation Attribute Warnings for the list of attributes that are not migrated. |
| Could not migrate SSL connector as no SSL config is defined | The message contains the explanation. |
| Could not migrate **verify-client** attribute *X* to the Undertow equivalent | The message contains the explanation. |
| Could not migrate **verify-client** expression *X* | The message contains the explanation. |

| Warning Message | What It Means / How to Fix It |
|---|---|
| Could not migrate valve *X* | The behavior exhibited by this valve in the previous release of JBoss EAP was not migrated. The administrator must verify if the new **undertow** subsystem in JBoss EAP 7 is able to operate correctly without that behavior or whether the behavior must be migrated manually.<br><br>This warning can occur for the following valves:<br><br>• **org.apache.catalina.valves.RemoteAddrValve**<br>It must have at least one allowed or denied value.<br>• **org.apache.catalina.valves.RemoteHostValve**<br>It must have at least one allowed or denied value.<br>• **org.apache.catalina.authenticator.BasicAuthenticator**<br>• **org.apache.catalina.authenticator.DigestAuthenticator**<br>• **org.apache.catalina.authenticator.FormAuthenticator**<br>• **org.apache.catalina.authenticator.SSLAuthenticator**<br>• **org.apache.catalina.authenticator.SpnegoAuthenticator**<br>• custom valves |

| Warning Message | What It Means / How to Fix It |
|---|---|
| Could not migrate attribute *X* from valve *Y* | The behavior exhibited by this valve attribute in the previous release of JBoss EAP was not migrated. The administrator must verify if the new **undertow** subsystem in JBoss EAP 7 is able to operate correctly without that behavior or whether the behavior must be migrated manually. This warning can occur for the following valve attributes:<br>• **org.apache.catalina.valves.AccessLogValve**<br>  ○ **resolveHosts**<br>  ○ **fileDateFormat**<br>  ○ **renameOnRotate**<br>  ○ **encoding**<br>  ○ **locale**<br>  ○ **requestAttributesEnabled**<br>  ○ **buffered**<br>• **org.apache.catalina.valves.ExtendedAccessLogValve**<br>  ○ **resolveHosts**<br>  ○ **fileDateFormat**<br>  ○ **renameOnRotate**<br>  ○ **encoding**<br>  ○ **locale**<br>  ○ **requestAttributesEnabled**<br>  ○ **buffered**<br>• **org.apache.catalina.valves.RemoteIpValve**<br>  ○ **httpServerPort**<br>  ○ **httpsServerPort**<br>  ○ **remoteIpHeader** If it is defined but not set to "x-forwarded-for"<br>  ○ **protocolHeader** If it is defined but not set to "x-forwarded-proto" |

### Web Subsystem Migration Operation Attribute Warnings

The **migrate** operation is not able to process all JBoss Web attributes. See the following reference tables for information about how to migrate the unprocessed attributes manually.

**Web SSL Connector Attributes**
The following attributes were used in JBoss EAP 6 to configure the SSL connector. OpenSSL native libraries are not supported in JBoss EAP 7 so there are no equivalent settings.

| Attribute | Description | Undertow Equivalent |
|---|---|---|
| ca-revocation-url | The file or URL that contains the revocation list. | No equivalent in Undertow. |
| certificate-file | When using OpenSSL encryption, the path to the file containing the server certificate. | No equivalent in Undertow. |
| ssl-protocol | The SSL protocol string. | No equivalent in Undertow. |
| verify-depth | The maximum number of intermediate certificate issuers checked before deciding that the clients do not have a valid certificate. | No equivalent in Undertow. |

**Web Static Resource Attributes**
The following **static-resources** element attributes were used to describe how static resources were handled by the **DefaultServlet** or by the **WebdavServlet**. There are no equivalents for these attributes because WebDAV is not supported by Undertow. For more information, see https://issues.jboss.org/browse/JBEAP-1036.

| Attribute | Description | Undertow Equivalent |
|---|---|---|
| disabled | Enable the default Servlet mapping. | No equivalent setting in Undertow. |
| file-encoding | File encoding to be used when reading static files. | No equivalent setting in Undertow. |
| max-depth | Maximum recursion for **PROPFIND**. | This is a WebDAV setting and WebDAV is not supported by Undertow. |
| read-only | Allow write HTTP methods (PUT, DELETE). | This is a WebDAV setting and WebDAV is not supported by Undertow. |
| secret | Secret for WebDAV locking operations. | This is a WebDAV setting and WebDAV is not supported by Undertow. |
| sendfile | Enable sendfile if possible, for files bigger than the specified byte size. | This is set to a sensible default value in Undertow and is not configurable. |
| webdav | Enable WebDAV functionality. | WebDAV is not supported by Undertow. |

### Web SSO Resource Attributes

SSO is handled differently than in the previous release and there are no equivalent attribute settings in JBoss EAP 7.

| JBoss Web Attribute | Description | Undertow Equivalent |
|---|---|---|
| cache-container | Name of the cache container to use for clustered SSO. | This setting is no longer needed in Undertow. This works by default across a distributed clustered environment. |
| cache-name | Name of the cache to use for clustered SSO. | This setting is no longer needed in Undertow. This works by default across a distributed clustered environment. |
| reauthenticate | Whether each request should cause a reauthentication. | There is no equivalent setting in Undertow, which behaves similarly to the **reauthenticate=true** setting in JBoss EAP 6. While **reauthenticate=false** could possibly improve performance, it could also create security issues. |

### Web Access Log Attributes

| JBoss Web Attribute | Description | Undertow Equivalent |
|---|---|---|
| resolve-hosts | Whether to enable resolving hosts for access logging. | Use the setting on the connector to accomplish the same behavior. |

### Web Connector Attributes

| JBoss Web Attribute | Description | Undertow Equivalent |
|---|---|---|
| executor | The name of the executor that should be used to process the threads of this connector. | You now reference a worker that is defined in the **io** subsystem.<br><br>See Migrate the Threads Subsystem Configuration for more information. |
| proxy-binding | The socket binding to define the host and port that is used when sending a redirect. | There is no direct equivalent.<br><br>See https-listener Attributes in the JBoss EAP *Configuration Guide* for available configuration options. |

| JBoss Web Attribute | Description | Undertow Equivalent |
|---|---|---|
| redirect-port | The port for redirection to a secure connector. | This attribute was deprecated in JBoss EAP 6 and replaced with **redirect-binding**. Undertow provides the **redirect-socket** attribute on the **http-listener** element, which is a replacement for **redirect-binding**.<br><br>See https-listener Attributes in the JBoss EAP *Configuration Guide* for more information. |

## A.4. MIGRATE JBOSS WEB SYSTEM PROPERTIES REFERENCE

This reference describes how to map system properties previously used for JBoss Web configuration to the equivalent configuration for Undertow in JBoss EAP 7.

- Map Servlet Container and Connectors System Properties
- Map EL System Properties
- Map JSP System Properties
- Map Security System Properties

Table A.1. Map Servlet Container and Connectors System Properties

| JBoss EAP 6 System Property | Description |
|---|---|
| | **Equivalent in JBoss EAP 7** |
| jvmRoute | Provides a default value for the **jvmRoute** attribute. It does not override the automatically generated value when using the **standalone-ha.xml** configuration file.<br><br>It supports **reload**. |
| | Management CLI command:<br><br>`/subsystem=undertow:write-attribute(name=instance-id,value=VALUE)` |
| org.apache.tomcat.util.buf.StringCache.byte.enabled | If **true**, the String cache is enabled for**ByteChunk**. If the value is not specified, the default value of **false** is used. |
| | No equivalent configuration |
| org.apache.tomcat.util.buf.StringCache.char.enabled | If **true**, the String cache is enabled for**CharChunk**. If the value is not specified, the default value of **false** is used. |

| | |
|---|---|
| | No equivalent configuration |
| org.apache.tomcat.util.buf.StringCache.cacheSize | The size of the String cache. If the value is not specified, the default value of **5000** is used. |
| | No equivalent configuration |
| org.apache.tomcat.util.buf.StringCache.maxStringSize | The maximum length of String that will be cached. If the value is not specified, the default value of **128** is used. |
| | No equivalent configuration |
| org.apache.tomcat.util.http.FastHttpDateFormat.CACHE_SIZE | The size of the cache to use parsed and formatted date value. If the value is not specified, the default value of **1000** is used. |
| | No equivalent configuration |
| org.apache.catalina.core.StandardService.DELAY_CONNECTOR_STARTUP | If **true**, the connector startup is not done automatically. It is useful in embedded mode. |
| | No equivalent configuration |
| org.apache.catalina.connector.Request.SESSION_ID_CHECK | If **true**, the Servlet container verifies that a session exists in a context with the specified session ID before creating a session with that ID. |
| | No equivalent configuration |
| org.apache.coyote.USE_CUSTOM_STATUS_MSG_IN_HEADER | If **true**, custom HTTP status messages are used within HTTP headers. Users must ensure that any such message is **ISO-8859-1** encoded, particularly if user provided input is included in the message, to prevent a possible XSS vulnerability. If value is not specified the default value of **false** is used. |
| | Must be enabled programmatically by implementing a custom **io.undertow.servlet.ServletExtension**. Then use the extension to call **setSendCustomReasonPhraseOnError(true)** on the **io.undertow.servlet.api.DeploymentInfo** structure instance. |
| org.apache.tomcat.util.http.Parameters.MAX_COUNT | The maximum number of parameters that can be parsed in a post body. If exceeded, parsing fails using an **IllegalStateException**. The default value is **512** parameters. |

| | |
|---|---|
| | Management CLI command:<br>`/subsystem=undertow/server=default-server/http-listener=default:write-attribute(name=max-parameters,value=VALUE)`<br>`/subsystem=undertow/server=default-server/https-listener=default:write-attribute(name=max-parameters,value=VALUE)`<br>`/subsystem=undertow/server=default-server/ajp-listener=default:write-attribute(name=max-parameters,value=VALUE)` |
| org.apache.tomcat.util.http.MimeHeaders.MAX_COUNT | The maximum number of headers that can be sent in the HTTP request. If exceeded, parsing will fail using an **IllegalStateException**. The default value is **128** headers. |
| | Management CLI command:<br>`/subsystem=undertow/server=default-server/http-listener=default:write-attribute(name=max-headers,value=VALUE)`<br>`/subsystem=undertow/server=default-server/https-listener=default:write-attribute(name=max-headers,value=VALUE)`<br>`/subsystem=undertow/server=default-server/ajp-listener=default:write-attribute(name=max-headers,value=VALUE)` |
| org.apache.tomcat.util.net.MAX_THREADS | The maximum number of threads a connector is going to use to process requests. The default value is **32** x **512**. (**512** x **Runtime.getRuntime().availableProcessors()** for the **JIO** connector) |
| | Management CLI command:<br>`/subsystem=io/worker=default:write-attribute(name=task-max-threads, value=VALUE)` |
| org.apache.coyote.http11.Http11Protocol.MAX_HEADER_SIZE | The maximum size of the HTTP headers, in bytes. If exceeded, parsing will fail using an **ArrayOutOfBoundsException**. The default value is **8192** bytes. |

| | |
|---|---|
| | Management CLI command:<br><br>`/subsystem=undertow/server=default-server/http-listener=default:write-attribute(name=max-header-size,value=VALUE)`<br>`/subsystem=undertow/server=default-server/https-listener=default:write-attribute(name=max-header-size,value=VALUE)`<br>`/subsystem=undertow/server=default-server/ajp-listener=default:write-attribute(name=max-header-size,value=VALUE)` |
| org.apache.coyote.http11.Http11Protocol.COMPRESSION | Allows using simple compression with the HTTP connector. The default value is **off**, and compression can be enabled using the value **on** to enable it conditionally, or force to always enable it. |
| | Configure a filter using the management CLI:<br><br>`# Create a filter`<br>`/subsystem=undertow/configuration=filter/gzip=gzipfilter:add()`<br>`/subsystem=undertow/server=default-server/host=default-host/filter-ref=gzipfilter:add()` |
| org.apache.coyote.http11.Http11Protocol.COMPRESSION_RESTRICTED_UA | User agents regexps that will not receive compressed content. The default value is empty. |
| | Configure a predicate in a filter using the management CLI:<br><br>`# Use a predicate in a filter`<br>`/subsystem=undertow/configuration=filter/gzip=gzipfilter:add()`<br>`/subsystem=undertow/server=default-server/host=default-host/filter-ref=gzipfilter:add(predicate="regex[pattern='AppleWebKit',value=%{i,User-Agent}]")` |
| org.apache.coyote.http11.Http11Protocol.COMPRESSION_MIME_TYPES | Content type prefixes of compressible content. The default value is **text/html,text/xml,text/plain**. |
| | Configure a predicate in a filter using the management CLI:<br><br>`# Use a predicate in a filter`<br>`/subsystem=undertow/configuration=filter/gzip=gzipfilter:add()`<br>`/subsystem=undertow/server=default-server/host=default-host/filter-ref=gzipfilter:add(predicate="regex[pattern='text/html',value=%{o,Content-Type}]")` |

| | | |
|---|---|---|
| org.apache.coyote.http11.Http11Protocol.COMPRESSION_MIN_SIZE | Minimum size of content that will be compressed. The default value is **2048** bytes. | Configure a predicate in a filter using the management CLI:<br>`# Use a predicate in a filter`<br>`/subsystem=undertow/configuration=filter/gzip=gzipfilter:add()`<br>`/subsystem=undertow/server=default-server/host=default-host/filter-ref=gzipfilter:add(predicate="max-content-size[value=MIN_SIZE]")` |
| org.apache.coyote.http11.DEFAULT_CONNECTION_TIMEOUT | Default socket timeout. The default value is **60000** ms. | Management CLI command:<br>`/subsystem=undertow/server=default-server/http-listener=default:write-attribute(name=no-request-timeout,value=VALUE)`<br>`/subsystem=undertow/server=default-server/https-listener=default:write-attribute(name=no-request-timeout,value=VALUE)`<br>`/subsystem=undertow/server=default-server/ajp-listener=default:write-attribute(name=no-request-timeout,value=VALUE)` |
| org.jboss.as.web.deployment.DELETE_WORK_DIR_ONCONTEXTDESTROY | Use this property to remove **.java** and **.class** files to ensure that JSP sources are recompiled. The default value is **false**. Default socket timeout for **keep-alive**. The default value is **-1** ms, which means it will use the default socket timeout. | No equivalent configuration |
| org.apache.tomcat.util.buf.StringCache.trainThreshold | Specifies the number of times **toString()** must be invoked before activating cache. The default value is **100000**. | No equivalent configuration |

**Table A.2. Map EL System Properties**

| JBoss EAP 6 System Property | Description | Equivalent in JBoss EAP 7 |
|---|---|---|

| | |
|---|---|
| org.apache.el.parser.COERCE_TO_ZERO | If **true**, when coercing expressions to numbers, empty strings ("") and null will be coerced to zero as required by the specification. If a value is not specified, the default value of **true** is used. |
| | System property is still valid and processed by the EL |

Table A.3. Map JSP System Properties

| JBoss EAP 6 System Property | Description |
|---|---|
| | **Equivalent in JBoss EAP 7** |
| org.apache.jasper.compiler.Generator.VAR_EXPRESSIONFACTORY | The name of the variable to use for the expression language expression factory. If value is not specified, the default value of **_el_expressionfactory** is used. |
| | System property has not changed |
| org.apache.jasper.compiler.Generator.VAR_INSTANCEMANAGER | The name of the variable to use for the instance manager factory. If value is not specified, the default value of **_jsp_instancemanager** is used. |
| | System property has not changed |
| org.apache.jasper.compiler.Parser.STRICT_QUOTE_ESCAPING | If **false**, the requirements for escaping quotes in JSP attributes are relaxed so that a missing required quote does not cause an error. If value is not specified, the specification compliant default of **true** is used. |
| | System property has not changed |
| org.apache.jasper.Constants.DEFAULT_TAG_BUFFER_SIZE | Any tag buffer that expands beyond **org.apache.jasper.Constants.DEFAULT_TAG_BUFFER_SIZE** is destroyed and a new buffer is created of the default size. If value is not specified, the default value of **512** is used. |
| | System property has not changed |
| org.apache.jasper.runtime.JspFactoryImpl.USE_POOL | If **true**, a ThreadLocal PageContext pool is used. If value is not specified, the default value of **true** is used. |
| | System property has not changed |
| org.apache.jasper.runtime.JspFactoryImpl.POOL_SIZE | The size of the ThreadLocal PageContext. If value is not specified, the default value of **8** is used. |
| | System property has not changed |

| | |
|---|---|
| org.apache.jasper.Constants.JSP_SERVLET_BASE | The base class of the Servlets generated from the JSPs. If value is not specified, the default value of **org.apache.jasper.runtime.HttpJspBase** is used. |
| | System property has not changed |
| org.apache.jasper.Constants.SERVICE_METHOD_NAME | The name of the service method called by the base class. If value is not specified, the default value of **_jspService** is used. |
| | System property has not changed |
| org.apache.jasper.Constants.SERVLET_CLASSPATH | The name of the ServletContext attribute that provides the class path for the JSP. If value is not specified, the default value of **org.apache.catalina.jsp_classpath** is used. |
| | System property has not changed |
| org.apache.jasper.Constants.JSP_FILE | The name of the request attribute for **<jsp-file>** element of a servlet definition. If present on a request, this overrides the value returned by **request.getServletPath()** to select the JSP page to be executed. If value is not specified, the default value of **org.apache.catalina.jsp_file** is used. |
| | System property has not changed |
| org.apache.jasper.Constants.PRECOMPILE | The name of the query parameter that causes the JSP engine to just pregenerate the servlet but not invoke it. If value is not specified, the default value of **org.apache.catalina.jsp_precompile** is used. |
| | System property has not changed |
| org.apache.jasper.Constants.JSP_PACKAGE_NAME | The default package name for compiled JSP pages. If value not specified, the default value of **org.apache.jsp** is used. |
| | System property has not changed |
| org.apache.jasper.Constants.TAG_FILE_PACKAGE_NAME | The default package name for tag handlers generated from tag files. If value is not specified, the default value of **org.apache.jsp.tag** is used. |
| | System property has not changed |
| org.apache.jasper.Constants.TEMP_VARIABLE_NAME_PREFIX | Prefix to use for generated temporary variable names. If value is not specified, the default value of **_jspx_temp** is used. |
| | System property has not changed |

| | |
|---|---|
| org.apache.jasper.Constants.USE_INSTANCE_MANAGER_FOR_TAGS | If **true**, the instance manager is used to obtain tag handler instances. If value is not specified, **true** is used. |
| | System property has not changed |
| org.apache.jasper.Constants.INJECT_TAGS | If **true**, annotations specified in tags will be processed and injected. This can have a performance impact when using simple tags, or if tag pooling is disabled. If value is not specified, **false** is used. |
| | System property has not changed |

Table A.4. Map Security System Properties

| JBoss EAP 6 System Property | Description |
|---|---|
| | **Equivalent in JBoss EAP 7** |
| org.apache.catalina.connector.RECYCLE_FACADES | If this is **true** or if a security manager is in use a new facade object is created for each request. If value is not specified, the default value of **false** is used. |
| | No equivalent configuration |
| org.apache.catalina.connector.CoyoteAdapter.ALLOW_BACKSLASH | If this is **true** the '\' character is permitted as a path delimiter. If value is not specified, the default value of **false** is used. |
| | No equivalent configuration |
| org.apache.tomcat.util.buf.UDecoder.ALLOW_ENCODED_SLASH | If this is **true**, '%2F' and '%5C' is permitted as path delimiters. If value is not specified, the default value of **false** is used. |
| | Management CLI command: `/subsystem=undertow/server=default-server/http-listener=default:write-attribute(name=allow-encoded-slash,value=VALUE)` `/subsystem=undertow/server=default-server/https-listener=default:write-attribute(name=allow-encoded-slash,value=VALUE)` `/subsystem=undertow/server=default-server/ajp-listener=default:write-attribute(name=allow-encoded-slash,value=VALUE)` |

| | |
|---|---|
| org.apache.catalina.STRICT_SERVLET_COMPLIANCE | If value is not specified, **true** is used. If this is **true** the following actions will occur: any wrapped request or response object passed to an application dispatcher is checked to ensure that it has wrapped the original request or response. (SRV.8.2 / SRV.14.2.5.1) a call to **Response.getWriter()** if no character encoding has been specified results in subsequent calls to **Response.getCharacterEncoding()** returning **ISO-8859-1** and the **Content-Type** response header will include a **charset=ISO-8859-1** component. (SRV.15.2.22.1) every request that is associated with a session causes the session's last accessed time to be updated regardless of whether or not the request explicity accesses the session. (SRV.7.6) |
| | Compliant by default |
| org.apache.catalina.core.StandardWrapperValve.SERVLET_STATS | If **true** or if org.apache.catalina.STRICT_SERVLET_COMPLIANCE is **true**, the wrapper will collect the JSR-77 statistics for individual servlets. If value is not specified, the default value of **false** is used. |
| | No equivalent configuration |
| org.apache.catalina.session.StandardSession.ACTIVITY_CHECK | If this is **true** or if **org.apache.catalina.STRICT_SERVLET_COMPLIANCE** is **true** Tomcat tracks the number of active requests for each session. When determining if a session is valid, any session with at least one active request is always be considered valid. If value is not specified, the default value of **false** is used. |
| | No equivalent configuration |

## A.5. COMPATIBILITY AND INTEROPERABILITY BETWEEN RELEASES

This section describes the compatibility and interoperability of client and server EJB and messaging components between the JBoss EAP 5, JBoss EAP 6, and JBoss EAP 7 releases.

### EJB remoting over IIOP

You should not encounter problems with any of the following configurations.

- Connecting from a JBoss EAP 5 client to a JBoss EAP 7 server
- Connecting from a JBoss EAP 6 client to a JBoss EAP 7 server
- Connecting from a JBoss EAP 7 client to a JBoss EAP 6 server
- Connecting from a JBoss EAP 7 client to a JBoss EAP 5 server

### EJB remoting Using JNDI

You should not encounter problems with any of the following configurations.

- Connecting from a JBoss EAP 6 client to a JBoss EAP 7 server

- Connecting from a JBoss EAP 7 client to a JBoss EAP 6 server

JBoss EAP 6 provided support for the EJB 3.1 specification and introduced the use of standardized global JNDI namespaces, which are still used in JBoss EAP 7. Due to the change in JNDI namespace names, the following configurations are not compatible:

- Connecting from a JBoss EAP 5 client to a JBoss EAP 7 or a JBoss EAP 6 server
- Connecting from a JBoss EAP 7 or JBoss EAP 6 client to a JBoss EAP 5 server

For more information about standardized JNDI namespace changes, see JNDI Changes in the JBoss EAP 6 *Migration Guide*.

#### EJB remoting Using @WebService

You should not encounter problems with any of the following configurations.

- Connecting from a JBoss EAP 5 client to a JBoss EAP 7 server
- Connecting from a JBoss EAP 6 client to a JBoss EAP 7 server
- Connecting from a JBoss EAP 7 client to a JBoss EAP 6 server
- Connecting from a JBoss EAP 7 client to a JBoss EAP 5 server

#### Messaging Standalone Client

You should not encounter problems with any of the following configurations.

- Connecting from a JBoss EAP 6 client to a JBoss EAP 7 server
- Connecting from a JBoss EAP 7 client to a JBoss EAP 6 server

In the following configuration, if the client is using the messaging broker-specific HornetQ API rather than the generic JMS API, the connection is possible. However, JNDI lookups must be addressed using the JBoss EAP legacy JNDI naming extension that is delivered with JBoss EAP 7.

- Connecting from a JBoss EAP 5 client to a JBoss EAP 7 server

JBoss EAP 7 built-in messaging is not able to connect to HornetQ 2.2.x that shipped with JBoss EAP 5 due to protocol compatibility issues. For this reason, the following configurations are not compatible.

- Connecting from a JBoss EAP 7 client to a JBoss EAP 5 server

#### Messaging MDBs

You should not encounter problems with any of the following configurations.

- Connecting from a JBoss EAP 6 client to a JBoss EAP 7 server
- Connecting from a JBoss EAP 7 client to a JBoss EAP 6 server

In the following configuration, if the client is using the messaging broker-specific HornetQ API rather than the generic JMS API, the connection is possible. However, JNDI lookups must be addressed using the JBoss EAP legacy JNDI naming extension that is delivered with JBoss EAP 7.

- Connecting from a JBoss EAP 5 client to a JBoss EAP 7 server

JBoss EAP 7 built-in messaging is not able to connect to HornetQ 2.2.x that shipped with JBoss EAP 5 due to protocol compatibility issues. For this reason, the following configurations are not compatible.

- Connecting from a JBoss EAP 7 client to a JBoss EAP 5 server

### JMS bridges

You should not encounter problems with any of the following configurations.

- Connecting from a JBoss EAP 5 client to a JBoss EAP 7 server
- Connecting from a JBoss EAP 6 client to a JBoss EAP 7 server
- Connecting from a JBoss EAP 7 client to a JBoss EAP 6 server
- Connecting from a JBoss EAP 7 client to a JBoss EAP 5 server

*Revised on 2019-09-26 10:41:40 UTC*