Red Hat JBoss Enterprise Application Platform 8.0

Migration Guide

Instructions for upgrading to a major version of Red Hat JBoss Enterprise Application Platform

Last Updated: 2024-02-21

Red Hat JBoss Enterprise Application Platform 8.0 Migration Guide

Instructions for upgrading to a major version of Red Hat JBoss Enterprise Application Platform

Legal Notice

Abstract

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

Table of Contents

PROVIDING FEEDBACK ON JBOSS EAP DOCUMENTATION

To report an error or to improve our documentation, log in to your Red Hat Jira account and submit an issue. If you do not have a Red Hat Jira account, then you will be prompted to create an account.

Procedure

1. Click the following link to **create a ticket**.
2. Enter a brief description of the issue in the **Summary**.
3. Provide a detailed description of the issue or enhancement in the **Description**. Include a URL to where the issue occurs in the documentation.
4. Clicking **Submit** creates and routes the issue to the appropriate documentation team.

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright’s message .

CHAPTER 1. RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM MIGRATION OVERVIEW

As a system administrator, you can upgrade from Red Hat JBoss Enterprise Application Platform 7 to Red Hat JBoss Enterprise Application Platform 8.0. This guide covers new features available in the release, deprecated and unsupported features, and any necessary application and server configuration updates to maintain consistent behavior.

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

After successfully deploying and running JBoss EAP 8.0, you can upgrade individual components to use the new functions and features of JBoss EAP 8.0.

NOTE

If you want to migrate from older releases of JBoss EAP, you must first migrate to JBoss EAP 7.4. For more information, see JBoss EAP 7.4 Migration Guide.

1.1. UNDERSTANDING MIGRATIONS AND UPGRADES

This section provides explanations and guidelines for upgrading and patching JBoss EAP, including major upgrades, minor updates, and cumulative patches.

1.1.1. Major upgrades in JBoss EAP

A major upgrade or migration is necessary when an application is moved from one major release to another, such as from JBoss EAP 7 to JBoss EAP 8.0. Applications that comply with the Jakarta EE 8 specification, contain proprietary code, or use deprecated APIs might require modifications to their application code to run on JBoss EAP 8.0. The introduction of Jakarta EE 10 involves changes in Java package names and other aspects that require adjustments to Jakarta EE application code for compatibility with JBoss EAP 8.0. Additionally, the server configuration has changed in JBoss EAP 8.0 and requires migration. This type of migration is addressed in this guide.

1.1.2. Minor updates in JBoss EAP

Minor Updates in JBoss EAP are point releases that provide bug fixes, security fixes, and new features. The changes made in each point release are documented in the Release notes for Red Hat JBoss Enterprise Application Platform 8.0.

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 more information, see Using the JBoss Server Migration Tool.

1.1.3. Cumulative patches in JBoss EAP

JBoss EAP periodically provides cumulative patches that contain bug and security fixes. Cumulative patches increment the release by the last digit, for example, upgrading from version 7.1.0 to 7.1.1.

1.2. USE OF *<EAP_HOME>* VARIABLE

In this document, the ***<EAP_HOME>*** variable denotes the path to the JBoss EAP installation. Replace this variable with the actual path to your JBoss EAP installation.

NOTE

<EAP_HOME> is not an environment variable. Use the **JBOSS_HOME** environment variable in scripts.

Depending on the installation option you choose to install JBoss EAP, you can locate the installation directory or the default path as follows:

- If you installed JBoss EAP by using the archive installation method, the installation directory is the **jboss-eap-8.0** directory where you extracted the archive.
- If you installed JBoss EAP by using the RPM installation method, the installation directory is the **/opt/rh/eap8/root/usr/share/wildfly/**.
- If you installed JBoss EAP by using the installer application, the default path for ***<EAP_HOME>*** is **${user.home}/EAP-8.0.0**:
  - For Red Hat Enterprise Linux and Oracle Solaris: **/home/*USER_NAME*/EAP-8.0.0/**
  - For Microsoft Windows: **C:\Users\*USER_NAME*\EAP-8.0.0\**
- If you installed and configured the JBoss EAP server by using the Red Hat CodeReady Studio installer application, the default path for ***<EAP_HOME>*** is **${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

If you set the **Target runtime** to **8.0** or a later runtime version in Red Hat CodeReady Studio, your project is compatible with the Jakarta EE 10 specification.

CHAPTER 2. PREPARING FOR MIGRATION TO JBOSS EAP 8.0

As a system administrator, you need to plan the migration to JBoss EAP 8.0. This upgrade is essential for improved performance, enhanced security, and increased stability of Java applications.

JBoss EAP 8.0 provides backward compatibility for JBoss EAP 7 applications. However, if your application uses features that JBoss EAP 8.0 has deprecated or removed, you might need to modify your application code.

The JBoss EAP 8.0 release introduces several changes that might impact your application deployment. To ensure a successful migration, conduct research and planning before attempting to migrate your application.

Before beginning the migration process, follow these initial steps:

- Familiarize yourself with the features of Jakarta EE 10 .
- Review features of JBoss EAP 8.0.
- Review the JBoss EAP getting started material .
- Ensure a seamless migration process by backing up your data and reviewing server state .
- Streamline your installation process by migrating JBoss EAP with RPM installation .
- Improve manageability and automation by migrating JBoss EAP as a service .

After becoming familiar with the feature changes, the development materials, and the tools that can assist your migration efforts, evaluate your applications and server configuration to determine the necessary changes for running them on JBoss EAP 8.0.

2.1. REVIEW THE JAKARTA EE 10 FEATURES

Jakarta EE 10 introduces numerous enhancements that simplify the development and deployment of feature-rich applications in both private and public clouds. It incorporates new features and the latest standards such as HTML5, WebSocket, JSON, Batch, and Concurrency Utilities. Updates include Jakarta Persistence 3.1, Jakarta RESTful Web Services 3.1, Jakarta Servlet 6.0, Jakarta Expression Language 5.0, Java Message Service 3.1. Jakarta Server Faces 4.0, Jakarta Enterprise Beans 4.0, Contexts and Dependency Injection 2.0, and Jakarta Bean Validation 3.0.

Additional resources

- Jakarta EE Platform 10

2.2. REVIEW THE FEATURES OF JBOSS EAP 8.0

JBoss EAP 8.0 includes upgrades and improvements over previous releases. For the complete list of new features introduced in JBoss EAP 8.0, see New features and enhancements in the *Release notes for Red Hat JBoss Enterprise Application Platform 8.0* on the Red Hat Customer Portal.

Before migrating your application to JBoss EAP 8.0, note that some features from previous releases may no longer be supported or have been deprecated due to high maintenance costs, low community interest, or availability of better alternatives. For a complete list of deprecated and unsupported features in JBoss EAP 8.0, see Unsupported, deprecated, and removed functionality in the *Release notes for Red Hat JBoss Enterprise Application Platform 8.0* on the Red Hat Customer Portal.

2.3. REVIEW THE JBOSS EAP GETTING STARTED MATERIAL

This section explains the key components of the JBoss EAP Getting Started material, providing a concise overview of essential information to help you start with JBoss EAP.

Review the JBoss EAP Getting Started Guide for essential information on:

- Downloading and installing JBoss EAP 8.0 to set up your environment effectively.
- Downloading and installing JBoss Tools to improve your development environment.

IMPORTANT

JBoss Tools is a community project and is not supported by Red Hat. Please reference the community website for assistance with setting up and running your instance of JBoss Tools. To download JBoss Tools, see JBoss Tools Downloads.

- Configuring Maven for your development environment and managing project dependencies.
- Downloading and running the quick-start example applications that come with the product.

Additional resources

- Developing applications using JBoss EAP

2.4. BACK UP YOUR DATA AND REVIEW SERVER STATE

This section emphasizes the need to back up data, review server state, and handle potential issues before migrating your application. By safeguarding deployments, managing open transactions, and assessing timer data, you can ensure a smooth migration.

Consider the following potential issues before you start the migration:

- The migration process might remove temporary folders. Make sure you backup any deployments within the **data/content/** directory before migrating. Later, restore the data after completion to avoid server failure due to missing content.
- Before migration, handle open transactions and delete the **data/tx-object-store/** transaction directory.
- Review the persistent timer data in **data/timer-service-data** before proceeding with the migration to determine its applicability post-upgrade. Before the upgrade, check the **deployment-*** files in that directory to identify which timers are still in use.

Make sure to back up the current server configuration and applications before you start the migration.

2.5. MIGRATE JBOSS EAP WITH RPM INSTALLATION

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 an archive or the **jboss-eap-installation-manager** 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. Therefore, it is recommended to migrate the JBoss EAP installation to a new machine when migrating to JBoss EAP 8.0.

Additional resources

- Installing JBoss EAP by using the RPM installation method

2.6. MIGRATE JBOSS EAP AS A SERVICE

If you run JBoss EAP 7 as a service, review the updated configuration instructions for JBoss EAP 8.0 in the Red Hat JBoss Enterprise Application Platform Installation Methods .

2.7. MIGRATE A CLUSTER

If you run a JBoss EAP cluster, follow the instruction in the Upgrading a cluster section in the JBoss EAP 7.4 *Patching and Upgrading Guide*.

CHAPTER 3. SIMPLIFY YOUR JBOSS EAP 8.0 MIGRATION WITH EFFECTIVE TOOLS

As a system administrator, you can simplify your migration process to JBoss EAP 8.0 with the help of two essential tools. The Migration Toolkit for Runtimes (MTR) analyzes your applications and provides detailed migration reports, whereas the JBoss Server Migration Tool updates your server configuration to include new features and settings.

3.1. ANALYZING YOUR APPLICATIONS BEFORE MIGRATION

You can use Migration Toolkit for Runtimes (MTR) to analyze the code and architecture of your JBoss EAP 6.4 and 7 applications before you migrate them to JBoss EAP 8.0. The MTR rule set for migration to JBoss EAP 8.0 provides reports on XML descriptors, specific application code, and parameters that need to be replaced by an alternative configuration when migrating to JBoss EAP 8.0.

MTR is an extensible and customizable rule-based set of tools that helps simplify migration of Java applications. MTR analyzes the APIs, technologies, and architectures used by the applications you plan to migrate, providing detailed migration reports for each application. These reports provide the following information:

- Detailed explanations of the necessary migration changes
- 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

Additional resources

- *Migration Toolkit for Runtimes*

3.2. SIMPLIFY YOUR SERVER CONFIGURATION MIGRATION

The JBoss Server Migration Tool is the preferred method for updating your server configuration to include the new features and settings in JBoss EAP 8.0 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 manage domains.

3.2.1. Migrating to JBoss EAP 8.0

The JBoss Server Migration Tool supports migration from all releases of JBoss EAP version 7, to JBoss EAP 8.0.

NOTE

If you want to migrate from JBoss EAP 6.4, you must first migrate to the latest Cumulative Patch (CP) version of JBoss EAP 7.4. For more information, see JBoss EAP 7.4 Migration Guide. Subsequently, you can migrate from JBoss EAP 7.4 CP version to JBoss EAP 8.0.

Prerequisites

- JBoss EAP is not running.

Procedure

1. Download the tool from the JBoss EAP download page .
2. Extract the downloaded archive.

   ```
   $ unzip <NAME_OF_THE_FILE>
   ```

3. Navigate to the **MIGRATION_TOOL_HOME/bin** directory.
4. Execute the **jboss-server-migration** script.
   - For Red Hat Enterprise Linux:

     ```
     $ ./jboss-server-migration.sh --source EAP_PREVIOUS_HOME --target
     EAP_NEW_HOME
     ```

   - For Microsoft Windows:

     ```
     jboss-server-migration.bat --source EAP_PREVIOUS_HOME --target EAP_NEW_HOME
     ```

     NOTE

     Replace **EAP_PREVIOUS_HOME** and **EAP_NEW_HOME** with the actual paths to the previous and new installations of JBoss EAP.

Additional resources

- Using the JBoss Server Migration Tool

CHAPTER 4. RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM APPLICATION MIGRATION FROM JAKARTA EE 8 TO 10

JBoss EAP 8.0 provides support for Jakarta EE 10. Jakarta EE 10 brings a large change to Jakarta EE compared to the Jakarta EE 8 specifications supported by JBoss EAP 7. In this chapter, the compatibility-impacting differences in the Jakarta EE APIs that application developers must be aware of when preparing to migrate their applications from JBoss EAP 7 to JBoss EAP 8.0 are discussed.

NOTE

The focus of this chapter is on the differences between Jakarta EE 8 and Jakarta EE 10 that an application developer migrating their application to JBoss EAP 8.0 might need to deal with, and not on how to do the migration. For more information on JBoss EAP 7 to JBoss EAP 8.0 application migration and the tools provided by Red Hat to assist with this, see Simplify your JBoss EAP 8.0 migration with effective tools and Understanding application migration changes.

4.1. THE JAVAX TO JAKARTA PACKAGE NAMESPACE CHANGE

By far the largest compatibility-impacting difference between Jakarta EE 8 and EE 10 is the renaming of the EE API Java packages from **javax.*** to **jakarta.***.

Following the move of Java EE to the Eclipse Foundation and the establishment of Jakarta EE, Eclipse and Oracle agreed that the Jakarta EE community cannot evolve the **javax.** package namespace. Therefore, in order to continue to evolve the EE APIs, beginning with Jakarta EE 9, the packages used for all EE APIs have changed from **javax.*** to **jakarta.***. This change does not affect javax packages that are part of Java SE.

Adapting to this namespace change is the biggest change involved in migrating an application from JBoss EAP 7 to JBoss EAP 8. Applications migrating to Jakarta EE 10 need to:

- Update any import statements or other source code uses of EE API classes from the **javax** package to **jakarta**
- Update the names of any EE-specified system properties or other configuration properties whose names that begin with **javax.** to instead begin with **jakarta.**
- Change the name of the resource that identifies the implementation class from **META-INF/services/javax.[rest_of_name]** to **META-INF/services/jakarta.[rest_of_name]** for any application-provided implementations of EE interfaces or abstract classes that are bootstrapped using the **java.util.ServiceLoader** mechanism.

NOTE

The Red Hat Migration Toolkit can assist in updating the namespaces in the application source code. For more information, see How to use Red Hat Migration Toolkit for Auto-Migration of an Application to the Jakarta EE 10 Namespace. For cases where source code migration is not an option, the open source Eclipse Transformer project provides bytecode transformation tooling to transform existing Java archives from the javax namespace to jakarta.

4.2. OTHER CHANGES

Besides the package namespace change, applications written for earlier EE versions may need to adapt to changes made in a number of specifications included in Jakarta EE 10. The following sections describe these changes, which are mostly removals of long-deprecated API elements.

In the following sections, for any instances of API elements that have been removed that use the **javax** namespace, the equivalent removal has been done in the **jakarta** namespace used in Jakarta EE 9. Therefore, if you have updated your application to replace the **javax** namespace with **jakarta**, assume that the items that mention **javax** are applicable for your application.

4.2.1. Jakarta Contexts and Dependency Injection Bean Discovery

As per the CDI 4.0 spec change notes, the default behavior for discovering Contexts and Dependency Injection or CDI beans in a deployment with an empty **beans.xml** file has changed from **all** to **annotated**. This means that for such a deployment only deployment classes with a bean defining annotation is discovered by CDI. If all application classes using beans have such an annotation, this CDI change will have no impact. Otherwise, an application deployment might fail when CDI cannot find a type that provides a particular bean.

If your application is impacted by this change, you have several options:

- Leave the **beans.xml** file empty but add a bean defining annotation to all classes that need it.
- Leave the classes unchanged but change the **beans.xml** file from being empty to one with the following content: **<beans bean-discovery-mode="all"></beans>**
- Leave the application unchanged, but change the server's weld subsystem configuration to restore handling of empty **beans.xml** files back to the JBoss EAP 7 behavior. This setting affects all deployments on the server. For example, with the CLI: **/subsystem=weld:write-attribute(name=legacy-empty-beans-xml-treatment,value=true)**

4.2.2. CDI API Changes

Jakarta Contexts and Dependency Injection 4.0 removed the following deprecated API elements:

- The **javax.enterprise.inject.spi.Bean.isNullable()** method has been removed. This method has always returned **false** for many years now, so applications that call it can replace the call with **false** or remove any branching logic and just retain the contents of the **false** branch.
- The **javax.enterprise.inject.spi.BeanManager.createInjectionTarget(AnnotatedType)** method has been removed. Replace this method call with with **BeanManager.getInjectionTargetFactory(AnnotatedType)** and use the returned factory to create injection targets. See Obtaining an InjectionTarget for a class in the Jakarta Contexts and Dependency injection specification for more information.
- The **javax.enterprise.inject.spi.BeanManager.fireEvent(Object, Annotation)** method has been removed. Use **BeanManager.getEvent()** as an entry point to a similar API. See Firing an event in the Jakarta Contexts and Dependency injection specification for more information.
- The **javax.enterprise.inject.spi.BeforeBeanDiscovery.addAnnotatedType(AnnotatedType)** method has been removed. If your application is calling this method, you can replace it with a call to **BeforeBeanDiscovery.addAnnotatedType(AnnotatedType, (String) null)**.

4.2.3. Jakarta Enterprise Beans

Java SE 14 has removed the **java.security.Identity** class, so it's usage has been removed from the Jakarta Enterprise Beans 4.0 API.

- The deprecated **javax.ejb.EJBContext.getCallerIdentity()** method has been removed. You can use **EJBContext.getCallerPrincipal()** instead, which returns **java.security.Principal**.
- The deprecated **javax.ejb.EJBContext.isCallerInRole(Identity role)** method has been removed. You can use **EJBContext.isCallerInRole(String roleName)** instead.
- The Jakarta XML RPC specification has been removed from the Jakarta EE 10 Full Platform, so the **javax.ejb.SessionContext.getMessageContext()** method that returned **javax.xml.rpc.handler.MessageContext** has been removed.
- The Jakarta XML RPC specification was optional in Jakarta EE 8, and Red Hat JBoss EAP 7 does not support it. Any usage of this specification would have thrown an **IllegalStateException**, so this EJB API change is not expected to affect any existing applications running on JBoss EAP 7.
- The deprecated **javax.ejb.EJBContext.getEnvironment()** method has been removed. Use the JNDI naming context **java:comp/env** to access the enterprise bean's environment.

4.2.4. Jakarta Expression Language

The incorrectly spelled **javax.el.MethodExpression.isParmetersProvided()** method has been removed. You can use **MethodExpression.isParametersProvided()** instead.

4.2.5. Jakarta JSON Binding

By default, types annotated with the **jakarta.json.bind.annotation.JsonbCreator** annotation does not require all parameters to be available in the JSON content. Default values will be used if the JSON being parsed is missing one of the parameters. The EE 8 behavior that requires all the parameters to be present in the JSON can be turned on by calling **jakarta.json.bind.JsonbConfig().withCreatorParametersRequired(true)**.

4.2.6. Jakarta Faces

The following deprecated functionality has been removed in Jakarta Faces 4.0.

4.2.6.1. Jakarta Faces and Java Server Pages

Jakarta Server Pages (JSP) support is deprecated in Jakarta Faces 2.0 and later versions. JSP support is removed in Jakarta Faces 4.0. Facelets replaces JSP as the preferred View Definition Language (VDL). Applications using JSP for Faces views can be modified using Facelets. You can identify the applications by mapping **FacesServlet** to the ***.jsp** suffix in **web.xml**.

4.2.6.2. Faces Managed-Beans

The deprecated Jakarta Faces-specific managed-bean concept has been removed in Faces 4.0, for Jakarta Contexts and Dependency Injection (CDI) beans. Applications using Faces managed-beans (i.e. classes annotated with **javax.faces.bean.ManagedBean** or referenced in a managed-bean element in **faces-config.xml**) might need to make the following changes:

- Classes annotated with **javax.faces.bean.ManagedBean** or referenced in a managed-bean element in **faces-config.xml** should instead be annotated with **jakarta.inject.Named**, and any managed-bean element in **faces-config.xml** should be removed.
- Members annotated with the **javax.faces.bean.ManagedProperty** annotation should use **jakarta.faces.annotation.ManagedProperty** instead, along with the **jakarta.inject.Inject**

annotation. To get a startup semantic similar to the old **javax.faces.bean.ManagedBean(name="foo", eager=true)**, add a **public void xxx(@Observes jakarta.enterprise.event.Startup event)** method or a **public void xxx(@Observes @Initialized(ApplicationScoped.class) Object context)** method. The **jakarta.enterprise.event.Startup** option is new in CDI 4.0.

- Use of the **javax.faces.bean.ApplicationScoped** annotation should be replaced with **jakarta.enterprise.context.ApplicationScoped**.
- Use of the **javax.faces.bean.CustomScoped** annotation should be replaced with CDI custom scopes and **jakarta.enterprise.context.spi.Context**. See Defining new scope types and The Context Interface in the CDI 4.0 specification for more details.
- Use of the **javax.faces.bean.NoneScoped** annotation should be replaced with **jakarta.enterprise.context.Dependent**, which is a CDI built-in scope with approximately similar semantics.
- Use of the **javax.faces.bean.RequestScoped** annotation should be replaced with **jakarta.enterprise.context.RequestScoped**.
- Use of the **javax.faces.bean.SessionScoped** annotation should be replaced with **jakarta.enterprise.context.SessionScoped**.

4.2.6.3. Other Faces API Changes

The **javax.faces.bean.ViewScoped** annotation has been removed. You can use **jakarta.faces.view.ViewScoped** instead.

The **javax.faces.view.facelets.ResourceResolver** and **javax.faces.view.facelets.FaceletsResourceResolver** annotations have been removed. For any ResourceResolvers in your application, implement the **jakarta.faces.application.ResourceHandler** interface and register the fully qualified class name of the implementation in the **application/resource-handler** element in **faces-config.xml**.

4.2.7. Jakarta Servlet

Jakarta Servlet 6.0 removes a number API classes and methods that were deprecated in Servlet 5.0 and earlier, mostly in the Servlet 2.x releases.

The **javax.servlet.SingleThreadModel** marker interface has been removed and servlets that implement this interface must remove the interface declaration and ensure that the servlet code properly guards state and other resource access against concurrent access. For example, by avoiding the usage of an instance variable or synchronizing the block of code accessing resources. However, it is recommended that developers do not synchronize the **service** method (or methods like **doGet** and **doPost** that it dispatches to) because of the detrimental effect of such synchronization on performance.

The **javax.servlet.http.HttpSessionContext** interface has been removed, along with the **javax.servlet.http.HttpSession.getSessionContext()** method. There have been no use cases for this interface since Servlet 2.1 as its implementations were required by specifications not to provide any usable data.

The **javax.servlet.http.HttpUtils** utility class has been removed. Applications should use the **ServletRequest** and **HttpServletRequest** interfaces instead of the following methods:

- **parseQueryString(String s)** and **parsePostData(int len, ServletInputStream in)** - Use **ServletRequest.getParameterMap()**. If an application needs to differentiate between query string parameters and request body parameters, the application must implement the code to do that by parsing the query string itself.

- **getRequestURL(HttpServletRequest req)**- Use **HttpServletRequest.getRequestURL()**.

Also, the following miscellaneous methods and constructors have been removed:

Class/Interface	Removed	Use Instead
javax.servlet.ServletContext	getServlet(String name)	no replacement
	getServlets()	no replacement
	getServletNames()	no replacement
	log(Exception exception, String msg)	log(String message, Throwable throwable)
javax.servlet.ServletRequest	getRealPath(String path)	ServletContext.getRealPath(String path)
javax.servlet.ServletRequestWrapper	getRealPath(String path)	ServletContext.getRealPath(String path)
javax.servlet.UnavailableException	getServlet()	no replacement
	UnavailableException(Servlet servlet, String msg)	UnavailableException(String)
	UnavailableException(int seconds, Servlet servlet, String msg)	UnavailableException(String, int)
javax.servlet.http.HttpServletRequest	isRequestedSessionIdFromUrl()	isRequestedSessionIdFromURL()

Class/Interface	Removed	Use Instead
javax.servlet.http.HttpServletRequestWrapper	isRequestedSessionIdFromUrl()	isRequestedSessionIdFromURL()
javax.servlet.http.HttpServletResponse	encodeUrl(String url)	encodeURL(String url)
	encodeRedirectUrl(String url)	encodeRedirectURL(String url)
	setStatus(int sc, String sm)	sendError(int, String)
javax.servlet.http.HttpServletResponseWrapper	encodeUrl(String url)	encodeURL(String url)
	encodeRedirectUrl(String url)	encodeRedirectURL(String url)
	setStatus(int sc, String sm)	sendError(int, String)
javax.servlet.http.HttpSession	getValue(String name)	getAttribute(String name)
	getValueNames()	getAttributeNames()
	putValue(String name, Object value)	setAttribute(String name, Object value)
	removeValue(String name)	removeAttribute(String name)

4.2.8. Jakarta Soap with Attachments

Support for provider lookup through a **jaxm.properties** file has been removed.

The deprecated **javax.xml.soap.SOAPElementFactory** class has been removed. Use **jakarta.xml.soap.SOAPFactory** for creating SOAPElements.

SOAPElementFactory method	SOAPFactory equivalent
newInstance()	newInstance()
create(Name)	createElement(Name)
create(String)	createElement(String)
create(String, String, String)	createElement(String, String, String)

4.2.9. Jakarta XML Binding

The XML namespace that should be used in xml binding files has changed. The **http://java.sun.com/xml/ns/jaxb** namespace should be replaced with **https://jakarta.ee/xml/ns/jaxb**.

The deprecated **javax.xml.bind.Validator** interface has been removed, as has the **associated javax.xml.bind.JAXBContext.createValidator()** method. To validate marshalling and unmarshalling operations, provide a **javax.xml.validation.Schema** to **jakarta.xml.bind.Marshaller.setSchema(Schema)**.

Support for compatibility with JAXB 1.0 has been removed.

Some of the deprecated steps in the **JAXBContext** implementation lookup algorithm have been removed. Searches for implementation class names through **jaxb.properties** files, **javax.xml.bind.context.factory** or **jakarta.xml.bind.JAXBContext** properties and **/META-INF/services/javax.xml.bind.JAXBContext** resource files have been dropped. For more informatoin about the current implementation discovery algorithm, see the Jakarta XML Binding 4.0 specification.

The generic requirements for a number of methods in the **javax.xml.bind.Marshaller** interface have changed as follows:

Jakarta XML Binding 2.3 / 3.0	Jakarta XML Binding 4.0
<A extends XmlAdapter> void setAdapter(A adapter)	<A extends XmlAdapter<?, ?>> void setAdapter(A adapter)
<A extends XmlAdapter> void setAdapter(Class<A> type, A adapter)	<A extends XmlAdapter<?, ?>> void setAdapter(Class<A> type, A adapter)
<A extends XmlAdapter> A getAdapter(Class<A> type)	<A extends XmlAdapter<?, ?>> A getAdapter(Class<A> type)

Apart from the changes in the Jakarta XML Binding API, there have been significant package name changes in the implementation library EAP 8, which might affect some applications that access the implementation library directly:

- Any use of classes in the **com.sun.xml.bind** package should be replaced by classes in the **org.glassfish.jaxb.runtime** package. Classes in sub-packages of **com.sun.xml.bind** should be replaced with classes in corresponding **org.glassfish.jaxb.runtime** sub-packages.

- For **jakarta.xml.bind.Marshaller** property settings, change the property constant name from **com.sun.xml.bind.*** to **org.glassfish.jaxb.***. For example, **marshaller.setProperty("com.sun.xml.bind.namespacePrefixMapper", mapper)** becomes **marshaller.setProperty("org.glassfish.jaxb.namespacePrefixMapper", mapper)**.

CHAPTER 5. MIGRATE A JBOSS EAP APPLICATION'S MAVEN PROJECT TO JBOSS EAP 8.0

When migrating an application's Maven project to JBoss EAP 8.0, which uses JBoss EAP BOMs to manage dependencies, you must update the pom.xml files due to the following significant changes introduced with the JBoss EAP 8.0 BOMs.

NOTE

If an application is migrated to JBoss EAP 8.0 without any changes, the application will build with incorrect dependencies and may fail to deploy on JBoss EAP 8.0.

5.1. RENAMING OF JBOSS EAP JAKARTA EE 8

The JBoss EAP **Jakarta EE 8** BOM has been renamed to **JBoss EAP EE** and its Maven Coordinates have been changed from **org.jboss.bom:jboss-eap-jakartaee8** to **org.jboss.bom:jboss-eap-ee**. The usage of this BOM in a Maven project (**pom.xml**) may be identified with the following dependency management import:

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.jboss.bom</groupId>
      <artifactId>jboss-eap-jakartaee8</artifactId>
      <version>${version.bom}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

The Maven project (**pom.xml**) should instead import the new "JBoss EAP EE" BOM to its dependency management:

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.jboss.bom</groupId>
      <artifactId>jboss-eap-ee</artifactId>
      <version>${version.bom}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

5.2. RENAMING OF JBOSS EAP JAKARTA EE 8 WITH TOOLS

The JBoss EAP **Jakarta EE 8 With Tools** BOM has been renamed to **JBoss EAP EE with tools**, and its Maven Coordinates have been changed from **org.jboss.bom:jboss-eap-jakartaee8-with-tools** to **org.jboss.bom:jboss-eap-ee-with-tools**. The usage of this BOM in a Maven project (**pom.xml**) may be identified with the following dependency management import:

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.jboss.bom</groupId>
      <artifactId>jboss-eap-jakartaee8-with-tools</artifactId>
      <version>${version.bom}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

The Maven project (**pom.xml**) must import the new “JBoss EAP EE With Tools” BOM to its dependency management instead:

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.jboss.bom</groupId>
      <artifactId>jboss-eap-ee-with-tools</artifactId>
      <version>${version.bom}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

5.3. REMOVAL OF JBOSS EAP JAKARTA EE 8 APIS

The JBoss EAP **Jakarta EE 8 APIs** BOMs have been removed in JBoss EAP 8.0 and the new **JBoss EAP EE** BOM must be used instead. The usage of this BOM in a Maven project (**pom.xml**) may be identified with the following dependency management import:

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.jboss.spec</groupId>
      <artifactId>jboss-jakartaee-8.0</artifactId>
      <version>${version.bom}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
    <dependency>
      <groupId>org.jboss.spec</groupId>
      <artifactId>jboss-jakartaee-web-8.0</artifactId>
      <version>${version.bom}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

The Maven project (**pom.xml**) must import the new **JBoss EAP EE** BOM to its dependency management:

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.jboss.bom</groupId>
      <artifactId>jboss-eap-ee</artifactId>
      <version>${version.bom}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

5.4. REMOVAL OF THE JBOSS EAP RUNTIME BOM

The JBoss EAP **Runtime** BOM is no longer distributed with JBoss EAP 8.0 and you must use the new **JBoss EAP EE** BOM instead. The usage of this BOM in a Maven project (**pom.xml**) may be identified with the following dependency management import:

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.jboss.bom</groupId>
      <artifactId>eap-runtime-artifacts</artifactId>
      <version>${version.bom}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

The Maven project (**pom.xml**) must import the new **JBoss EAP EE** BOM to its dependency management instead:

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.jboss.bom</groupId>
      <artifactId>jboss-eap-ee</artifactId>
      <version>${version.bom}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

5.5. JAKARTA EE AND JBOSS APIS MAVEN COORDINATES CHANGES

The following Jakarta EE and JBoss APIs artifacts, which were provided by the JBoss EAP BOMs, have changed Maven Coordinates, or were replaced by other artifacts:

- **com.sun.activation:jakarta.activation**
- **org.jboss.spec.javax.annotation:jboss-annotations-api_1.3_spec**

- **org.jboss.spec.javax.security.auth.message:jboss-jaspi-api_1.0_spec**
- **org.jboss.spec.javax.security.jacc:jboss-jacc-api_1.5_spec**
- **org.jboss.spec.javax.batch:jboss-batch-api_1.0_spec**
- **org.jboss.spec.javax.ejb:jboss-ejb-api_3.2_spec**
- **org.jboss.spec.javax.el:jboss-el-api_3.0_spec**
- **org.jboss.spec.javax.enterprise.concurrent:jboss-concurrency-api_1.0_spec**
- **org.jboss.spec.javax.faces:jboss-jsf-api_2.3_spec**
- **org.jboss.spec.javax.interceptor:jboss-interceptors-api_1.2_spec**
- **org.jboss.spec.javax.jms:jboss-jms-api_2.0_spec**
- **com.sun.mail:jakarta.mail**
- **org.jboss.spec.javax.resource:jboss-connector-api_1.7_spec**
- **org.jboss.spec.javax.servlet:jboss-servlet-api_4.0_spec**
- **org.jboss.spec.javax.servlet.jsp:jboss-jsp-api_2.3_spec**
- **org.apache.taglibs:taglibs-standard-spec**
- **org.jboss.spec.javax.transaction:jboss-transaction-api_1.3_spec**
- **org.jboss.spec.javax.xml.bind:jboss-jaxb-api_2.3_spec**
- **org.jboss.spec.javax.xml.ws:jboss-jaxws-api_2.3_spec**
- **javax.jws:jsr181-api**
- **org.jboss.spec.javax.websocket:jboss-websocket-api_1.1_spec**
- **org.jboss.spec.javax.ws.rs:jboss-jaxrs-api_2.1_spec**
- **org.jboss.spec.javax.xml.soap:jboss-saaj-api_1.4_spec**
- **org.hibernate:hibernate-core**
- **org.hibernate:hibernate-jpamodelgen**
- **org.jboss.narayana.xts:jbossxts**

The Maven Project (**pom.xml**) should update the dependency's Maven Coordinates if the artifact was changed or replaced, or remove the dependency if the artifact is no longer supported.

```
<dependencies>
  <!-- replaces com.sun.activation:jakarta.activation -->
  <dependency>
    <groupId>jakarta.activation</groupId>
    <artifactId>jakarta.activation-api</artifactId>
  </dependency>
  <!-- replaces org.jboss.spec.javax.annotation:jboss-annotations-api_1.3_spec -->
```

```
<dependency>
  <groupId>jakarta.annotation</groupId>
  <artifactId>jakarta.annotation-api</artifactId>
</dependency>
<!-- replaces org.jboss.spec.javax.security.auth.message:jboss-jaspi-api_1.0_spec -->
<dependency>
  <groupId>jakarta.authentication</groupId>
  <artifactId>jakarta.authentication-api</artifactId>
</dependency>
<!-- replaces org.jboss.spec.javax.security.jacc:jboss-jacc-api_1.5_spec -->
<dependency>
  <groupId>jakarta.authorization</groupId>
  <artifactId>jakarta.authorization-api</artifactId>
</dependency>
<!-- replaces org.jboss.spec.javax.batch:jboss-batch-api_1.0_spec -->
<dependency>
  <groupId>jakarta.batch</groupId>
  <artifactId>jakarta.batch-api</artifactId>
</dependency>
<!-- replaces org.jboss.spec.javax.ejb:jboss-ejb-api_3.2_spec -->
<dependency>
  <groupId>jakarta.ejb</groupId>
  <artifactId>jakarta.ejb-api</artifactId>
</dependency>
<!-- replaces org.jboss.spec.javax.el:jboss-el-api_3.0_spec -->
<dependency>
  <groupId>org.jboss.spec.jakarta.el</groupId>
  <artifactId>jboss-el-api_5.0_spec</artifactId>
</dependency>
<!-- replaces org.jboss.spec.javax.enterprise.concurrent:jboss-concurrency-api_1.0_spec -->
<dependency>
  <groupId>jakarta.enterprise.concurrent</groupId>
  <artifactId>jakarta.enterprise.concurrent-api</artifactId>
</dependency>
<!-- replaces org.jboss.spec.javax.faces:jboss-jsf-api_2.3_spec -->
<dependency>
  <groupId>jakarta.faces</groupId>
  <artifactId>jakarta.faces-api</artifactId>
</dependency>
<!-- replaces org.jboss.spec.javax.interceptor:jboss-interceptors-api_1.2_spec -->
<dependency>
  <groupId>jakarta.interceptor</groupId>
  <artifactId>jakarta.interceptor-api</artifactId>
</dependency>
<!-- replaces org.jboss.spec.javax.jms:jboss-jms-api_2.0_spec  -->
<dependency>
  <groupId>jakarta.jms</groupId>
  <artifactId>jakarta.jms-api</artifactId>
</dependency>
<!-- replaces com.sun.mail:jakarta.mail -->
<dependency>
  <groupId>jakarta.mail</groupId>
  <artifactId>jakarta.mail-api</artifactId>
</dependency>
<!-- replaces org.jboss.spec.javax.resource:jboss-connector-api_1.7_spec -->
<dependency>
```

```
        <groupId>jakarta.resource</groupId>
        <artifactId>jakarta.resource-api</artifactId>
    </dependency>
    <!-- replaces org.jboss.spec.javax.servlet:jboss-servlet-api_4.0_spec -->
    <dependency>
        <groupId>jakarta.servlet</groupId>
        <artifactId>jakarta.servlet-api</artifactId>
    </dependency>
    <!-- replaces org.jboss.spec.javax.servlet.jsp:jboss-jsp-api_2.3_spec -->
    <dependency>
        <groupId>jakarta.servlet.jsp</groupId>
        <artifactId>jakarta.servlet.jsp-api</artifactId>
    </dependency>
    <!-- replaces org.apache.taglibs:taglibs-standard-spec -->
    <dependency>
        <groupId>jakarta.servlet.jsp.jstl</groupId>
        <artifactId>jakarta.servlet.jsp.jstl-api</artifactId>
    </dependency>
    <!-- replaces org.jboss.spec.javax.transaction:jboss-transaction-api_1.3_spec  -->
    <dependency>
        <groupId>jakarta.transaction</groupId>
        <artifactId>jakarta.transaction-api</artifactId>
    </dependency>
    <!-- replaces org.jboss.spec.javax.xml.bind:jboss-jaxb-api_2.3_spec  -->
    <dependency>
        <groupId>jakarta.xml.bind</groupId>
        <artifactId>jakarta.xml.bind-api</artifactId>
    </dependency>
    <!-- replaces org.jboss.spec.javax.xml.ws:jboss-jaxws-api_2.3_spec and javax.jws:jsr181-api -->
    <dependency>
        <groupId>org.jboss.spec.jakarta.xml.ws</groupId>
        <artifactId>jboss-jakarta-xml-ws-api_4.0_spec</artifactId>
    </dependency>
    <!-- replaces org.jboss.spec.javax.websocket:jboss-websocket-api_1.1_spec -->
    <dependency>
        <groupId>jakarta.websocket</groupId>
        <artifactId>jakarta.websocket-api</artifactId>
    </dependency>
    <!-- replaces org.jboss.spec.javax.ws.rs:jboss-jaxrs-api_2.1_spec -->
    <dependency>
        <groupId>jakarta.ws.rs</groupId>
        <artifactId>jakarta.ws.rs-api</artifactId>
    </dependency>
    <!-- replaces org.jboss.spec.javax.xml.soap:jboss-saaj-api_1.4_spec -->
    <dependency>
        <groupId>org.jboss.spec.jakarta.xml.soap</groupId>
        <artifactId>jboss-saaj-api_3.0_spec</artifactId>
    </dependency>
    <!-- replaces org.hibernate:hibernate-core -->
    <dependency>
        <groupId>org.hibernate.orm</groupId>
        <artifactId>hibernate-core</artifactId>
    </dependency>
    <!-- replaces org.hibernate:hibernate-jpamodelgen -->
    <dependency>
        <groupId>org.hibernate.orm</groupId>
        <artifactId>hibernate-jpamodelgen</artifactId>
    </dependency>
    <!-- replaces org.jboss.narayana.xts:jbossxts -->
    <dependency>
        <groupId>org.jboss.narayana.xts</groupId>
        <artifactId>jbossxts-jakarta</artifactId>
        <classifier>api</classifier>
    </dependency>
</dependencies>
```

5.6. REMOVAL OF JBOSS EJB CLIENT LEGACY BOM

The JBoss EJB Client Legacy BOM, with Maven **groupId:artifactId** coordinates **org.jboss.eap:wildfly-ejb-client-legacy-bom**, is no longer provided with JBoss EAP.

You can identify this BOM used as dependency management import or as dependency reference in a Maven project **pom.xml** configuration file as illustrated in the following examples:

Example dependency management import

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.jboss.bom</groupId>
      <artifactId>wildfly-ejb-client-legacy-bom</artifactId>
      <version>${version.bom}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

Example dependency reference

```
<dependencies>
...
  <dependency>
    <groupId>org.jboss.bom</groupId>
    <artifactId>wildfly-ejb-client-legacy-bom</artifactId>
    <version>${version.bom}</version>
    <type>pom</type>
  </dependency>
...
</dependencies>
```

Stand-alone client Java applications can continue to use the same version of the removed BOM, for example, version **7.4.0.GA**, to work with JBoss EAP 8. However, it is recommended to replace the EJB Client Legacy API. For information about configuring an EJB Client, see How configure an EJB client in EAP 7.1+ / 8.0+. For information about deprecation of EJB Client Legacy API in JBoss EAP 7.4, see Deprecated in Red Hat JBoss Enterprise Application Platform Platform (EAP) 7 .

IMPORTANT

The JBoss EAP 7.4 client and JBoss EAP 8.x server follow different product lifecycle. For more information, see Product lifecycle for JBoss EAP.

CHAPTER 6. SERVER MIGRATION CHANGES

Before migrating, ensure you understand the migration changes necessary for deploying applications on a server and upgrading them in Red Hat JBoss Enterprise Application Platform 8.0.

6.1. WEB SERVER CONFIGURATION CHANGES

Learn about changes in **mod_cluster** and Undertow within Red Hat JBoss Enterprise Application Platform that impact root context behavior and enhance the security of your server information.

6.1.1. Default web module behavior changes

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
```

Additional resources

- Configure the Default Welcome Web Application

6.1.2. Undertow subsystem default configuration changes

Prior to Red Hat JBoss Enterprise Application Platform 7.2, the default **undertow** subsystem configuration included two response header filters that were appended to each HTTP response by the **default-host**:

- **Server** was previously set to **JBoss EAP/7**.
- **X-Powered-By** was previously 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 default **undertow** subsystem configuration in JBoss EAP 7.4.

```
<subsystem xmlns="urn:jboss:domain:undertow:12.0" 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>
```

The following is an example of the default **undertow** subsystem configuration in JBoss EAP 8.0.

```
<subsystem xmlns="urn:jboss:domain:undertow:14.0" default-virtual-host="default-host" default-
servlet-container="default" default-server="default-server" statistics-
enabled="${wildfly.undertow.statistics-enabled:${wildfly.statistics-enabled:false}}" default-security-
domain="other">
    <byte-buffer-pool name="default"/>
```

```
  <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" ssl-context="applicationSSC" enable-
http2="true"/>
    <host name="default-host" alias="localhost">
      <location name="/" handler="welcome-content"/>
      <http-invoker http-authentication-factory="application-http-authentication"/>
    </host>
  </server>
  <servlet-container name="default">
    <jsp-config/>
    <websockets/>
  </servlet-container>
  <handlers>
    <file name="welcome-content" path="${jboss.home.dir}/welcome-content"/>
  </handlers>
  <application-security-domains>
    <application-security-domain name="other" security-domain="ApplicationDomain"/>
  </application-security-domains>
</subsystem>
```

6.2. INFINISPAN SERVER CONFIGURATION CHANGES

Configure a custom stateful session bean (SFSB) cache for passivation in Red Hat JBoss Enterprise Application Platform 7.1 and later while considering the following aspects:

- Deprecation of the idle-timeout attribute
- Implementation of lazy passivation
- Determination of cluster name
- Appropriate configuration of eviction and expiration
- Modifications in the cache container transport protocol for enhanced performance.

By adhering to these considerations, you can optimize your SFSB cache configuration for improved passivation in JBoss EAP 7.1 and beyond.

6.2.1. Configuring custom stateful session bean cache for passivation

In JBoss EAP 7.1 and later versions, a custom stateful session beans (SFSB) cache with passivation enabled has changed. When configuring SFSB cache with passivation, consider the following key changes:

- Deprecation of the idle-timeout attribute
- A shift from eager to lazy passivation
- Determining the cluster name
- Configuring eviction and expiration in the Jakarta Enterprise Beans cache

When configuring a custom SFSB cache for passivation in JBoss EAP 7.1 and later versions, consider the following restrictions:

- 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 7.1 and later only support lazy passivation, which occurs when the **max-size** threshold is reached.

  NOTE

  Eager passivation through idle-timeout is no longer supported in these versions.

- In JBoss EAP 7.1 and later, the cluster name used by the Jakarta Enterprise Beans 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.

6.2.2. Infinispan cache container transport changes

A behavior change between JBoss EAP 7.0 and later versions requires performing updates to the cache container transport protocol in batch mode or using a special header. This change also affects tools used for managing 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.

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

6.2.3. EJB subsystem configuration changes from version 8.0 and later

JBoss EAP 8.0 introduces changes to the Enterprise JavaBeans (EJB) subsystem configuration for distributable stateful session beans (SFSB), including a new subsystem and updates to several resources. Several resources used in JBoss EAP 6 and 7 are also deprecated. These changes enable server configuration migration to ensure that your applications are compatible with future major releases.

JBoss EAP 8.0 replaces the deprecated resources used in JBoss EAP 6 and 7 with two new resources and a **distributable-ejb** subsystem for configuring SFSB caching distributively. The following table outlines the deprecated resources and the new resources that replace them.

Table 6.1. SFSB cache configuration changes

Deprecated resources	New non-distributable SFSB cache	New distributable SFSB cache
/subsystem=ejb3/cache	**/subsystem=ejb3/simple-cache**	**/subsystem=ejb3/distributable-cache**
/subsystem=ejb3/passivation-store	NA	**/subsystem=ejb3/distributable-cache=”name”/bean-management"=..**

Non-distributable SFSB cache, **/subsystem=ejb3/simple-cache**, is equivalent to the SFSB cache, **/subsystem=ejb3/cache**, used in JBoss EAP 7, where no passivation store was defined.

Distributable SFSB cache, **/subsystem=ejb3/distributable-cache**, includes an optional **bean-management** attribute that refers to a corresponding resource from the **distributable-ejb** subsystem. If you do not define the resource, it defaults to the **bean-management** resource within the **distributable-ejb** subsystem.

Consider migrating your server configuration to the updated approach in JBoss EAP 8.0. Although the current release continues to function with the deprecated resources, this might not be the case with future releases when they get removed.

An example of a comparison between JBoss EAP 7 and preferred JBoss EAP 8.0 configurations is as follows:

JBoss EAP 7 configuration:

```
/subsystem=ejb3/cache=example-simple-cache:add()
/subsystem=ejb3/passivation-store=infinispan:add(cache-container=ejb, bean-cache=default, max-size=1024)
/subsystem=ejb3/cache=example-distributed-cache:add(passivation-store=infinispan)
```

Preferred JBoss EAP 8.0 configuration:

```
/subsystem=ejb3/simple-cache=example-simple-cache:add()
/subsystem=distributable-ejb=example-distributed-cache/infinispan-bean-management=example-bean-cache:add(cache-container=ejb, cache=default, max-active-beans=1024)
/subsystem=ejb3/distributable-cache=example-distributed-cache:add(bean-management=example-bean-cache)
```

Adopting the preferred JBoss EAP 8.0 configuration ensures that your servers are compatible with the latest version and future major releases. You will also benefit from improved resources and subsystems for distributable SFSBs.

6.3. JAKARTA ENTERPRISE BEANS SERVER CONFIGURATION CHANGES

While configuring the **ejb3** subsystem in JBoss EAP 7, exceptions may appear in the server log during deployment of enterprise bean applications.

IMPORTANT

If you use the JBoss Server Migration Tool to update your server configuration, ensure that the **ejb3** subsystem is properly configured and no issues arise when deploying your Jakarta Enterprise Beans applications. For information about configuring and running the tool, see Using the JBoss Server Migration Tool .

6.3.1. Resolving DuplicateServiceException due to caching changes

The following **DuplicateServiceException** error 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
```

To resolve the **DuplicateServiceException** caused by caching changes in JBoss EAP 7, run 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])
```

By reconfiguring the cache, you can resolve this error and prevent the **DuplicateServiceException** from occurring.

6.4. MESSAGING SERVER CONFIGURATION CHANGES

Learn how to migrate both your configuration and associated messaging data to ActiveMQ Artemis, which serves as the Jakarta Messaging support provider in Red Hat JBoss Enterprise Application Platform 8.0.

6.4.1. Migrate messaging data

Review the approaches you can take to migrate messaging data in Red Hat JBoss Enterprise Application Platform.

To migrate messaging data from a previous JBoss EAP 7.x release to JBoss EAP 8.0, you can Migrate messaging data by using export and import approaches. This method involves exporting messaging data from the previous release and importing it into JBoss EAP 8.0 using the management CLI **import-journal** operation. Note that this approach is specifically applicable to file-based messaging systems.

As with version 7, JBoss EAP 8.0 continues to use ActiveMQ Artemis as the Jakarta Messaging support provider, which helps to make the migration process smoother.

6.4.1.1. Migrate messaging data by using export and import approaches

Use the following approach to export the messaging data from a previous release to an XML file, and then import that file using the **import-journal** operation:

1. Export messaging data from JBoss EAP 7.x. release
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.

6.4.1.1.1. Export messaging data from JBoss EAP 7.x release

To export messaging data from Red Hat JBoss Enterprise Application Platform 7.x release, follow the outlined procedure.

Prerequisites

- JBoss EAP 7.x is installed on your system.
- You have access to a terminal or command line interface.
- You have the necessary permissions to navigate directories and execute commands.

Procedure

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()
   ```

Verification

- Make sure there are no errors or warning messages in the log at the completion of the command.
- Use tool compatible with your operating system to validate the XML in the generated output file.

6.4.1.1.2. Import the XML formatted messaging data

After exporting messaging data from a JBoss EAP 8.0, you need to import the XML file into JBoss EAP 8.0 or later using the **import-journal** operation.

Prerequisites

- Complete the migration of your JBoss EAP 8.0 by using either the management CLI migrate operation or the JBoss Server Migration Tool.
- Start the JBoss EAP 8.0 server in normal mode without any connected Jakarta Messaging clients.

Procedure

To import the XML file into JBoss EAP 8.0 or a later version, follow these steps using the **import-journal** operation:

> **IMPORTANT**
>
> If your target server has already performed some messaging tasks, make 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. For more information, see Backing up messaging folder data.

1. Start the JBoss EAP 8.0 server in normal mode with *no* Jakarta Messaging clients connected.

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

2. Open a new terminal, navigate to the JBoss EAP 8.0 install directory, and connect to the management CLI.

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

3. 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.

6.4.1.1.3. Recovering from an import messaging data failure

You can recover from an import messaging data failure if the **import-journal** operation fails.

Prerequisites

Prerequisites

- Familiarity with the JBoss EAP 8.0 server and its management CLI commands.
- Knowledge of the directory location of messaging journal folders.
- Prior backup of target server messaging data if available.

Procedure

1. Shut down the JBoss EAP 8.0 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.

6.4.1.2. Migrate messaging data using a messaging bridge

A Jakarta Messaging bridge consumes messages from a source Jakarta Messaging queue or topic and sends them to a target Jakarta Messaging queue or topic, located on a different server. It enables message bridging between messaging servers that adhere to the Jakarta Messaging 3.1 standards. Look up the source and destination Jakarta Messaging resources using Java Naming and Directory Interface, ensuring that the client classes for Java Naming and Directory Interface lookup are bundled in a module and declare the module name in the Jakarta Messaging bridge configuration.

This section provides instructions on how to configure the servers and deploy a messaging bridge for moving messaging data from JBoss EAP 7 to JBoss EAP 8.0. To achieve this, proceed with the following steps:

1. Configuring JBoss EAP 8.0 server
2. Migrating the messaging data

6.4.1.2.1. Configuring JBoss EAP 8.0 server

To configure the Jakarta Messaging bridge in JBoss EAP 8.0 for seamless migration of messaging data, including module dependencies and queue configuration, follow the outlined procedure.

Prerequisites

- JBoss EAP 8.0 server installed and running.

Procedure

1. Create the following **jms-queue** configuration for the default server in the **messaging-activemq** subsystem of the JBoss EAP 8.0 server.

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

2. 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])
```

3. Create and deploy a Jakarta Messaging bridge that reads messages from the **InQueue** JMS queue 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"=>"http-remoting://legacy-host:8080")],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 8.0 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">
    <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="http-remoting://legacy-host:8080"/>
        </source-context>
    </source>
    <target destination="jms/queue/MigratedMessagesQueue" connection-
factory="java:/ConnectionFactory"/>
</jms-bridge>
```

6.4.1.2.2. Migrating the messaging data

To migrate messaging data from Red Hat JBoss Enterprise Application Platform 8.0 to Red Hat JBoss Enterprise Application Platform 8.0, follow the outlined procedure.

Prerequisites

- JBoss EAP 8.0 server installed and running.

Procedure

1. Verify that the information you provided for the following configurations is correct.
   - Any queue and topic names.
   - The **java.naming.provider.url** for Java Naming and Directory Interface lookup.
2. Make sure that you have deployed the target Jakarta Messaging destination to the JBoss EAP 8.0 server.
3. Start the JBoss EAP 8.0 servers, including the JBoss EAP 7 servers involved in the migration process.

6.4.1.3. Backing up messaging folder data

To ensure data integrity, it is recommended to back up the target message folders before making any changes if your server has already processed messages. You can find the default location of the messaging folders at ***EAP_HOME*/standalone/data/activemq/**; however, it might be configurable. If you are unsure about the location of your messaging data, you can use the following management CLI commands to determine it.

Procedure

1. Determine the location of your messaging data by using the following management CLI commands:

```
/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
```

   NOTE

   Ensure that you stop the server before copying the data.

2. Copy each messaging folder to a secure backup location after you identify their respective locations.

6.4.2. Configure the Jakarta Messaging resource adapter

The way you configure a generic Jakarta Messaging resource adapter for use with a third-party Jakarta Messaging provider has changed in Red Hat JBoss Enterprise Application Platform 8.0. For more information, see Deploying a generic Java Message Service resource adapter in the JBoss EAP 7.4 *Configuring Messaging* guide.

6.4.3. Messaging configuration changes

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

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

```
/subsystem=messaging-activemq/server=default/ha-policy=replication-primary: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-primary: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"}
}
```

6.4.4. Galleon layer for embedded broker messaging

In JBoss EAP 7, an embedded messaging broker was part of the default installation. In JBoss EAP 8, this functionality was added to a new Galleon layer called as **embedded-activemq**.

This new layer is not a part of the default configuration so users who want to rely on having a broker embedded in JBoss EAP must include it explicitly in their configuration.

The layer provides a **messaging-activemq** subsystem with an embedded broker even if it is recommended for customers to use a dedicated AMQ cluster on OpenShift. It also provisions ancillary resources, for example, socket-bindings and necessary dependencies needed to support this use case.

6.5. SECURITY ENHANCEMENTS IN JBOSS EAP 8.0

Starting with JBoss EAP 8.0, you must use Elytron since the legacy security subsystem and legacy security realms are no longer available. You can only configure Elytron defaults by using the JBoss Server Migration Tool. Therefore, legacy security configurations must be manually migrated.

Additional resources

- Migrating to Elytron

6.5.1. Vaults migration

Vaults has been removed from JBoss EAP 8.0. Use the credential store provided by the **elytron** subsystem to store sensitive strings.

Additional resources

- Migrate secure vaults and properties
- Credentials and credential stores in Elytron

6.5.2. Legacy security subsystem and security realms removal

The legacy security subsystem and legacy security realms have been removed from JBoss EAP 8.0. Use the security realms provided by the **elytron** subsystem.

Additional resources

- Migrate Authentication Configuration
- Migrate Database Authentication Configuration to Elytron
- Migrate Composite Stores to Elytron
- Migrate Security Domains That Use Caching to Elytron
- Migrating Legacy Properties-based Configuration to Elytron
- Migrating LDAP Authentication Configuration to Elytron
- Migrate Kerberos Authentication to Elytron
- Migrate SSL Configurations
- Securing applications and management interfaces using an identity store
- Securing applications and management interfaces using multiple identity stores

6.5.3. PicketLink subsystem removal

The PicketLink subsystem has been removed from JBoss EAP 8.0. Use Red Hat build of Keycloak instead of the PicketLink identity provider, and the Red Hat build of Keycloak SAML adapter instead of the PicketLink service provider.

Additional resources

- PicketLink removal
- Red Hat build of Keycloak

6.5.4. Migrate from Red Hat build of Keycloak OIDC client adapter to JBoss EAP subsystem

The **keycloak** subsystem is not supported in JBoss EAP 8.0 and is replaced by the **elytron-oidc-client** subsystem. JBoss Server Migration Tool performs the migration by default.

Additional resources

- Migrate **keycloak** subsystem
- OpenID Connect configuration in JBoss EAP

6.5.5. Custom login modules migration

In JBoss EAP 8.0, the legacy security subsystem has been removed. To continue using your custom login modules with the **elytron** subsystem, use the new Java Authentication and Authorization Service (JAAS) security realm and **jaas-realm**.

Additional resources

- JAAS realm in the **elytron** subsystem

6.5.6. FIPS mode changes

Starting from JBoss EAP 7.1, automatic generation of a self-signed certificate is enabled by default for development purposes. If you are running in FIPS mode, configure the server to disable automatic self-signed certificate creation. Failure to do so may lead to the following error upon starting 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
```

Additional resources

- Enabling SSL/TLS for applications by using the automatically generated self-signed certificate

6.6. MOD_CLUSTER CONFIGURATION CHANGES

The configuration for static proxy lists in mod_cluster has changed in Red Hat JBoss Enterprise Application Platform 7.4.

Starting from JBoss EAP 7.4, the **proxy-list** attribute was deprecated and subsequently removed in JBoss EAP 8.0.

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 7.4 *Configuration Guide*.

To ensure compatibility with JBoss EAP 8.0, update user scripts and legacy user CLI script as follows:

- Replace the deprecated **ssl=configuration** with the appropriate **elytron-based** configuration.
- Update the mod_cluster configuration path from **/mod-cluster-config=CONFIGURATION** to **/proxy=default**.
- Update the dynamic load provider path in user scripts, replacing the deprecated path with **provider=dynamic**.
- The deprecated **connector** attribute, which referred to an Undertow listener, has been removed. Update your user scripts to use the **listener** attribute as a replacement.

For more information about mod_cluster attributes, see ModCluster subsystem attributes in the JBoss EAP 7.4 *Configuration Guide*.

6.7. VIEWING CONFIGURATION CHANGES

With Red Hat JBoss Enterprise Application Platform 7, you can track the configuration changes that were made to a running server. You can also view the history of configuration changes made by authorized users.

Whereas with JBoss EAP 7.0, you had to use the **core-service** management CLI command to configure options and to retrieve a list of 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 7.4 *Configuration Guide*.

CHAPTER 7. UNDERSTANDING APPLICATION MIGRATION CHANGES

This section describes the changes required for migrating an application from JBoss EAP 6.4 or 7.x to JBoss EAP 8.0.

7.1. 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. JBoss EAP 8.0 then uses JBossWS 7 to support its features.

7.1.1. 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. Starting with JBoss EAP 7, it is no longer supported. Applications that use JAX-RPC must be migrated to use Jakarta XML Web Services, which is the current Jakarta 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.

7.1.2. 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 8. 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).

7.1.2.1. 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:5.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:jakartaee="https://jakarta.ee/xml/ns/jakartaee"
  xsi:schemaLocation="urn:jboss:jbossws-jaxws-config:5.0 schema/jbossws-jaxws-config_5_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>
```

7.1.2.2. 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:5.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xmlns:jakartaee="https://jakarta.ee/xml/ns/jakartaee"
  xsi:schemaLocation="urn:jboss:jbossws-jaxws-config:5.0 schema/jbossws-jaxws-config_5_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>
```

7.1.2.3. 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.

7.1.3. WS-Security changes

This section describes the various WS-Security changes for your application in JBoss EAP 6.4 and JBoss EAP 7.0.

- 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.
- Usage of 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.

7.1.4. 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.

7.1.5. 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.

Starting with JBoss EAP 7, it was included with the **org.bouncycastle** module and JBossWS was 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 1.72 or greater.

7.1.6. 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.

7.1.7. Jakarta XML Web Services 2.2 requirements for WebServiceRef

Containers must use Jakarta XML Web Services 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. Starting with JBoss EAP 7 that included JBossWS 5, this requirement is not hidden. This states that user provided service classes injected by the container must implement Jakarta XML Web Services 2.2 or later by updating the existing code to use the **jakarta.xml.ws.Service** constructor that includes one or more **WebServiceFeature** arguments.

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

7.1.8. 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**.

7.1.9. 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.

7.1.10. 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.

Starting with the JBoss EAP 7 release, if your application used the JBossWS implementation of Apache CXF, it ensured that 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.

7.1.11. Specification of descriptor in EAR archive

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

Starting with JBoss EAP 7, you can 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 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.

7.2. UPDATE THE REMOTE URL CONNECTOR AND PORT

Starting with JBoss EAP 7, the default connector has been 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**.

7.3. MESSAGING APPLICATION CHANGES

This section describes the various messaging application changes in JBoss EAP 7. In addition, you can learn more about how to:

- Change Jakarta Messaging deployment descriptors
- Update external Jakarta Messaging clients
- Replace deprecated address setting attributes
- Configure the required messaging application changes

7.3.1. Replace or update Jakarta Messaging deployment descriptors

Starting with JBoss EAP 7, the proprietary HornetQ messaging resource deployment descriptor files identified by the naming pattern **-jms.xml** does not work. The following is an example of a Java Message Service 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** Java Message Service deployment descriptors in your application in the previous release, you can either convert your application to use the standard deployment descriptor as specified in the Resource Definition and Configuration section of the Jakarta EE platform, 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>
```

7.3.2. 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.

7.3.3. 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 since JBoss EAP 8.0 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
auto-create-jms-topics	auto-create-addresses
auto-delete-jms-topics	auto-delete-addresses

In JBoss EAP 6, the default address setting attributes were set to **false**. Starting with JBoss EAP 7, the replacement attributes are set to **true** by default.

If you prefer to preserve the JBoss EAP 6 behavior, you must set the replacement attributes to **false**.

For more information about these replacement attributes, see Address Setting Attributes in the JBoss EAP 7.4 *Configuring Messaging Guide*.

7.3.4. Messaging application changes required for JBoss EAP 7

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.

7.4. JAKARTA RESTFUL WEB SERVICES AND RESTEASY APPLICATION CHANGES

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

JBoss EAP 7 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.

JBoss EAP 7.4 includes RESTEasy 3.15, which is an implementation of Jakarta RESTful Web Services 2.1. 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 to RESTEasy 3.15. For more information about the Java API for RESTEasy RESTEasy 3.15, see RESTEasy Jakarta RESTful Web Services 3.15.0.Final API.

JBoss EAP 8.0 provides support for RESTEasy 6.2, which implements the Jakarta RESTful Web Services 3.1 specification.

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 Jakarta RESTful Web Services.

7.4.1. 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 Jakarta REST compliant interceptor facility in RESTEasy 3.x. The relevant interfaces are defined in the **jakarta.ws.rs.ext** package of the **jakarta.ws.rs.api** module.

The following providers have been removed in JBoss EAP 8.0:

- **org.jboss.resteasy:resteasy-jackson-provider**
- **org.jboss.resteasy:resteasy-jettison-provider**
- **org.jboss.resteasy:resteasy-yaml-provider**

The following has been removed in JBoss EAP 8.0 as they now have Jakarta RESTful Web Services replacements.

- **@Suspend** and **org.jboss.resteasy.spi.AsynchronousResponse** have been removed and replaced with **@Suspended** and **javax.ws.rs.container.AsyncResponse** respectively.
- **StringConverter** is replaced by a **ParamConverter**.
- **org.jboss.resteasy.plugins.providers.SerializableProvider** was deprecated and has been removed.
- The following interceptor interfaces deprecated in RESTEasy 3.x has been removed.
  - **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 **jakarta.ws.rs.container.ContainerRequestFilter** interface in RESTEasy 3.x.
- The following interfaces and classes have been removed from RESTEasy 3.x and JBoss EAP 8.0.
  - **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 **jakarta.ws.rs.ext.WriterInterceptor** interface.
- In addition, some changes to the **jakarta.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 **jakarta.ws.rs.Produces** and the **@Produces** annotation as follows.

```
...
import jakarta.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 Jakarta REST filter and interceptor interfaces.

Additional resources

- RESTEasy Interceptors
- RESTEasy Jakarta RESTful Web Services 3.15.0.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 have been removed from JBoss EAP 8.0.
  - **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 **jakarta.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()));
jakarta.ws.rs.core.Link link = response.getLink("previous");
```

See the **resteasy-jaxrs-client** quickstart for an example of an external Jakarta REST RESTEasy client that interacts with a Jakarta REST 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.annotations.cache** package.

NOTE

For more information about the **org.jboss.resteasy.client.jaxrs** API classes, see the RESTEasy Jakarta REST JavaDoc .

StringConverter

The **org.jboss.resteasy.spi.StringConverter** class is deprecated in RESTEasy 3.x and JBoss EAP 8.0. This functionality can be replaced using the Jakarta REST jakarta.ws.rs.ext.ParamConverterProvider class.

7.4.2. 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 Jakarta REST method should be cached on the server, was removed from RESTEasy 3 and must be removed from the application code.

7.4.3. Additional RESTEasy changes

This section provides information about some additional changes in RESTEasy for JBoss EAP.

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.
- You can use **SignedOutput** and **SignedInput** 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 JBoss EAP 7.4 *Developing Web Services Applications*.

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 JBoss EAP 7.4 *Developing Web Services Applications*.

SerializableProvider
Deserializing Java objects from untrusted sources is not safe. For this reason, starting with 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();
   }
}
```

7.4.4. RESTEasy SPI changes

The RESTEasy SPI provider has been removed in JBoss EAP 8.

SPI Exceptions
All SPI failure exceptions were deprecated and are no longer used internally. They have been replaced with the corresponding Jakarta REST exception.

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

7.4.5. Jackson provider changes

The version of Jackson included in JBoss EAP 6.4 has changed. Starting with JBoss EAP 7, 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**.

7.4.6. Spring RESTEasy integration changes

JBoss EAP 8.0 provides support for RESTEasy 6.2. If you plan to use the Spring 6.0 framework with JBoss EAP 8.0, you must use Java 17.

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.

7.4.7. RESTEasy Jettison JSON provider changes

The RESTEasy Jettison JSON provider is deprecated since 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 7.4 *Development Guide*.

7.4.8. MicroProfile for JBoss EAP

MicroProfile is the name of a specification that developers can use to configure applications and microservices to run in multiple environments without having to modify or repackage those apps. Previously, *MicroProfile* was available for JBoss EAP 7.3 as a technology preview, but it has since been removed. *MicroProfile* is now available only on JBoss EAP XP.

Additional resources

- Understand MicroProfile

7.5. CDI APPLICATION CHANGES

JBoss EAP 8.0 includes support for CDI 4.0. As a result, applications written using older CDI releases might see some changes in behavior when migrating to JBoss EAP 8.0. This section summarizes only a few of these changes.

For more information about Weld and CDI 4.0, see:

- Jakarta Context Dependency Injection 4.0
- Weld 5.1.1.Final - CDI Reference Implementation

7.5.1. Bean Archives

Bean classes of enabled beans must be deployed in bean archives to ensure they are discovered by CDI and processed as beans.

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. The Jakarta equivalents for bean defining annotations are defined in the Jakarta Context Dependency Injection 2.0 specification.

In CDI 4.0:

- An archive does not differentiate whether beans.xml has a version number.
- In addition to build-compatible extensions, an archive also contains archives without the beans.xml file. The build compatible extensions are not bean archives.
- The default discovery mode of an archive with an empty beans.xml file is set to **annotated** instead of **all**. For example, if the **beans.xml** file is empty, it is an implicit bean archive instead of an explicit bean archive.
- In both cases, the bean discovery element is unaffected between archives with and without **beans.xml** files.

For more information about CDI 4.0, see Jakarta Contexts and Dependency Injection 4.0.

A bean archive has a bean discovery mode of **all**, **annotated** or **none**. A bean archive which contains non-empty beans.xml 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 portable extension or a build compatible extension and no **beans.xml** file.

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

- The archive contains a **beans.xml** file with **bean-discovery-mode** of **all**.

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

- The archive contains a beans.xml file that is empty.
- 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.

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

7.5.2. 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 Jakarta EE .

7.5.3. 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.

7.6. 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 and EAP 8 returns only the session ID.

This change can create issues with routeless cookies for some upgrades from JBoss EAP 6 to JBoss EAP 8. 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.

7.7. 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 or are deprecated in 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>
```

7.8. HIBERNATE CHANGES

JBoss EAP 8 includes support for Hibernate ORM 6.2, an object-relational mapping tool for the Java programming language. For more information about the Hibernate ORM 6.2 documentation, see Hibernate ORM 6.2.

When migrating from JBoss EAP 7.4 to JBoss EAP EAP 8.0, refer to the specific Hibernate ORM migration documentation for your Hibernate ORM version.

- For migrating from JBoss EAP 7.4 to JBoss EAP 8, you must complete the following steps.
  - Migrating from Hibernate ORM 5.3 to 5.4
  - Migrating from Hibernate ORM 5.4 to 5.5
  - Migrating from Hibernate ORM 5.5 to 5.6
  - Migrating from Hibernate ORM 5.6 to 6.0
  - Migrating from Hibernate ORM 6.0 to 6.1
  - Migrating from Hibernate ORM 6.1 to 6.2
  - Hibernate ORM dialects
  - Deprecated Hibernate ORM classes
  - Incubating Hibernate ORM classes
  - Hibernate ORM internals
- For migrating from older versions of JBoss EAP and Hibernate, you must complete the following steps.
  - Migrating from Hibernate ORM 4.3 to Hibernate ORM 5.0

   - Migrating from Hibernate ORM 5.0 to Hibernate ORM 5.1
   - Migrating from Hibernate ORM 5.1 and Hibernate ORM 5.2 to Hibernate ORM 5.3

Additional resources

- Hibernate ORM 6.2 Dialects
- Deprecated Hibernate ORM classes
- Incubating Hibernate ORM classes
- Hibernate ORM internals

7.8.1. Migrating from Hibernate ORM 5.3 to 5.4

This section highlights the changes required when migrating from Hibernate ORM version 5.3 to 5.4. For more information about the changes implemented between Hibernate ORM 5.3 and Hibernate ORM 5.4, see the Hibernate ORM 5.4 Migration Guide.

Known Changes
The following describes some of the changes when migrating from Hibernate ORM version 5.3 to 5.4.

7.8.1.1. Overriding Delayed Identity Insert Behavior

- In Hibernate 5.3, support was provided for **DelayedPostInsertIdentifier** behavior to be influenced based on the **FlushMode** or **FlushModeType** values, in short enhancing **Extended PersistenceContext** support. Unfortunately, there were a few issues that were included in this change.
- In Hibernate 5.4, it was decided to preserve as much of the Hibernate 5.3 behavior as possible and only restore very specific **DelayedPostInsertIdentifier** behavior for selected use cases.
- In order to make Hibernate 5.4 more flexible, a configuration option was added to be used as a temporary solution to completely disable the Hibernate 5.3 behavior, reverting it back to Hibernate 5.2 and earlier.

7.8.1.2. SQL Server JDBC Driver version upgrade to at least 6.1.2

Due to fixing HHH-12973, you must upgrade the JDBC Driver version to 6.1.2. Due to this issue, the older versions of the SQL Server JDBC Driver cannot introspect the **INFORMATION_SCHEMA.SEQUENCES** without closing the database connection.

7.8.2. Migrating from Hibernate ORM 5.4 to 5.5

This section highlights the changes required when migrating from Hibernate ORM version 5.4 to 5.5. For more information about the changes implemented between Hibernate ORM 5.4 and Hibernate ORM 5.5, see the Hibernate ORM 5.5 Migration Guide.

Known Changes
The Hibernate ORM 5.5 version is similar to Hibernate ORM 5.4 as it includes all bugfixes applied to the 5.4 maintenance releases and introduces support for Jakarta Persistence API.

7.8.2.1. Dom4J based XML mapping

The implementation of Hibernate’s parsing of XML mapping definitions has been completley reworked based on JAXB rather than DOM4J, to ensure continous progress for removing this dependecy.

7.8.2.2. Removed the ability to disable "enhanced proxies"

The "enhanced proxies" feature had been introduced as an optional performance improvement feature for Hibernate 5.3. This feature is now enabled permanently.

7.8.3. Migrating from Hibernate ORM 5.5 to 5.6

This section highlights the changes required when migrating from Hibernate ORM version 5.5 to 5.6. For more information about the changes implemented between Hibernate ORM 5.5 and Hibernate ORM 5.6, see the Hibernate ORM 5.6 Migration Guide.

Deprecated features
The Hibernate 5.6 version is very similar to the previous Hibernate 5.5 version, with the exception of removal of some of the deprecated features from previous Hibernate releases.

7.8.3.1. Removal of Javassist

You can no longer choose **javassist** as an implementation to be used for the bytecode enhancement of entities. **Byte Buddy** is the default, and **javassist** has been deprecated for some time and now removed. This does not have any functional impact on applications; the only exception being that it’s not longer valid to configure the **hibernate.bytecode.provider=javassist property**. You can remove this property if you are using this feautre. It can cause an issue where Hibenate ORM no longer lists **javassist** among its dependencies.

7.8.4. Migrating from Hibernate ORM 5.6 to 6.0

This section highlights the changes required when migrating from Hibernate ORM version 5.6 to 6.0. For more information about the changes implemented between Hibernate ORM 5.6 and Hibernate ORM 6.0, see the Hibernate ORM 6.0 Migration Guide.

The Hibernate 6.0 release includes the following changes:

- Java 11 is the minimum compatible baseline version for Hibernate 6.0.
- Jakarta Persistence: Another important change in the Hibernate ORM 6.0 release includes moving from the Java Persistence as defined by the Java EE specs to Jakarta Persistence as defined by the Jakarta EE spec. The most important impact resulting from this change includes the use of the Jakarata Persistence classes **jakarta.persistence.*** instead of the Java Persistence ones **javax.persistence.***.
- Reading from JDBC: Another reason for the development of the Hibernate ORM 6.0 version was to move from reading results from the JDBC ResultSet by name (read-by-name) to reading the results by position (read-by-position). This change was made to improve the scaling by undertaking throughput testing.
- Generated SQL: This feature resulted in the following enhancements:
  - Column aliases are no longer generated
  - Column references are "unique-d".
  - Better definition of joins and better determination of unnecessary joins (secondary tables, inheritance tables)

- Identifier as Object - The earlier versions of Hibernate required that all identifier types implement **Serializable**, Hibernate 6.0 has removed this restriction as identifiers can be any **Object**. This change affects many API and SPI methods previously defined using **Serializable**.

- @IdGeneratorType: With this release, you can use the **@IdGeneratorType** annotation for better type-safe way to define custom generators for identifier generation.

- Implicit Identifier Sequence and Table Name: The method by which Hibernate determines implicit names for sequences and tables associated with identifier generation has been modified in Hibernate 6.0, This may affect the user migrating applications. In this release, Hibernate creates a sequence per entity hierarchy instead of a single sequence **hibernate_sequence** by default.

- Defaults for implicit sequence generators: Implicit sequences, like the **hibernate_sequence** earlier now adhere to the default value of the JPA **@SequenceGenerator** annotation, which means that the sequences have an allocation size of 50.

- Type system: As Hibernate 6.0 is a major release, another important change was to modify Hibernate's mapping annotations and make them more type-safe. This feature was decided to be provided in this release as type-related contracts were already changing to implement the read-by-position changes.

- Query: A lot of changes have been introduced in the functionality for Query. Query features such as moving to a dedicated tree structure to model HQL and Criteria queries, improving the implementations for bulk SQM DML statements like insert, update, and delete, as well as changing the behavior of the **hibernate.criteria.copy_tree** property, and the inclusion of pass-through tokens have been introduced in this release.

- Signature change of the **#onSave** method: The signature of the **#onSave** method has been changed from **boolean onSave(Object entity, Serializable id, Object[] state, String[] propertyNames, Type[] types)** to **boolean onSave(Object entity, Object id, Object[] state, String[] propertyNames, Type[] types)** to accomadate the change of expected identifier type from **Serializable** to **Object**.

- Fetch circularity determination: Previous versions of Hibernate determined fetches using a depth-first approach, which occasionally led to odd "circularity" determination. Starting with Hibernate 6.0, now fetch determination is performed using a width first approach.

- Restructuring of org.hibernate.loader: The contents of the **loader.collection** package were restructured into **loader.ast.spi** and **loader.ast.internal** as well as adapted to the SQM API.

- Restructuring of the SQL package: The contents of **sql.ordering** were moved to **metamodel.mapping.ordering.ast**.

- Deprecation of hbm.xml mappings: Legacy **hbm.xml** mapping format is deprecated and will no longer supported beyond 6.x.

- Lazy association adherance: Prior to Hibernate 6.0, lazy associations that used **fetch="join" or @Fetch(FetchMode.JOIN)** were considered eager when loaded by id i.e. through **Session#get**/**EntityManager#find**, even though for queries the association was treated as lazy. Starting with Hibernate 6.0, the laziness of such associations is properly respected, regardless of the fetch mechanism. Backwards compatibility can be achieved by specifying **lazy="false" or @ManyToOne(fetch = EAGER)**/**@OneToOne(fetch = EAGER)**/**@OneToMany(fetch = EAGER)**/**@ManyToMany(fetch = EAGER)**.

- Change of behavior for **hbm.xml <return-join/>**: As per Hibernate 6.0, a **<return-join/>**` cause an association to be fetched, rather than adding a selection item.

For more information about these features, see Hibernate ORM 6.0 Migration Guide.

The Hibernate 6.0 release contains also contains lot of feature removals from previous Hibernate releases, which are listed as follows:

- **hbm.xml** multiple **<column/>** is now disallowed - In 6.0 the support for basic property mappings with multiple columns was removed. The component class attribute now supports interpreting a **CompositeUserType** class properly.
- Legacy Hibernate Criteria API - The legacy Hibernate Criteria API which was deprecated back in Hibernate 5.x has been removed in Hibernate 6.0.
- Callable via NativeQuery - Using **NativeQuery** to call SQL functions and procedures is no longer supported. Use methods such as **org.hibernate.procedure.ProcedureCall** or **jakarta.persistence.StoredProcedureQuery** instead.
- HQL fetch all properties clause - The fetch all properties clause was removed from the HQL language.
- JMX integration - Hibernate no longer provides built-in support for integrating itself with JMX environments.
- JACC integration - Hibernate no longer provides built-in support for integrating itself with JACC environments.

For more information about features removed in Hibernate 6.0, see Hibernate ORM 6.0 Migration Guide.

7.8.5. Migrating from Hibernate ORM 6.0 to 6.1

This section highlights the changes required when migrating from Hibernate ORM version 6.0 to 6.1. For more information about the changes implemented between Hibernate ORM 6.0 and Hibernate ORM 6.1, see the Hibernate ORM 6.1 Migration Guide .

The Hibernate 6.1 release contains the following changes:

- Basic arrays: Basic arrays other than **byte[]**/**Byte[]** and **char[]**/**Character[]**, and basic collections (only subtypes of Collection) now map to the type code **SqlTypes.ARRAY** by default, which maps to the SQL standard array type as determined by the new methods **getArrayTypeName** and **supportsStandardArrays** of **org.hibernate.dialect.Dialect**.
- Enum mapping changes: Enums now map to the type code **SqlType.SMALLINT** by default, as before it mapped to **TINYINT**. This mapping was not quite correct as Java effectively allows up to 32K enum entries, but **TINYINT** is only a 1 byte type.

For more detailed information about features included in the Hibernate 6.1, see the Hibernate ORM 6.1 Migration Guide.

7.8.6. Migrating from Hibernate ORM 6.1 to 6.2

This section highlights the changes required when migrating from Hibernate ORM version 6.1 to 6.2. For more information about the changes implemented between Hibernate ORM 6.1 and Hibernate ORM 6.2, see the Hibernate ORM 6.2 Migration Guide .

The Hibernate 6.2 release contains the following enhancements, which are listed as follows:

DDL type changes:

- OffsetTime mapping changes - In this release, **OffsetTime** depends on **@TimeZoneStorage** and the **hibernate.timezone.default_storage** setting. As the default setting is **TimeZoneStorageType.DEFAULT**, it means that the DDL expectations for such columns have changed.
- UUID mapping changes on MariaDB - On MariaDB, the type code **SqlTypes.UUID** by default refers to the DDL type **uuid**, as compared to before where it was using **binary(16)**. Due to this change, schema validation errors can occur on existing databases.
- UUID mapping changes on SQL Server - On SQL Server, the type code **SqlTypes.UUID** by default refers to the DDL type **uniqueidentifier**, as compared to before where it was using **binary(16)**. Due to this change, schema validation errors can occur on existing databases.
- JSON mapping changes on Oracle - On Oracle 12.1+, the type code **SqlTypes.JSON** by default refers to the DDL type blob and on 21+ to json, as compared to before where it was using **clob**. Due to this change, schema validation errors can occur on existing databases.
- JSON mapping changes on H2 - On H2 1.4.200+, the type code **SqlTypes.JSON** by default refers to the DDL type JSON, as compared to before where it was using **clob**. Due to this change, schema validation errors can occur on existing databases.
- Datatype for enums - Starting from Hibernate 6.2, the choice of implicit SQL datatype for storing enums is sensitive to the number of entries defined on the enum class.
- Timezone and offset storage - **hibernate.timezone.default_storage** now defaults to **DEFAULT**.
- Byte[]/Character[] mapping changes - Hibernate 6.2 makes it configurable to handle mapping of **Byte[]** and **Character[]** mapping changes in a domain model.
- UNIQUE constraint for optional one-to-one mappings - Earlier versions of Hibernate did not create a **UNIQUE** constraint on the database for logical one-to-one associations marked as **optional**. Starting in Hibernate 6.2, those UNIQUE constraints are now created.
- Column type inference for number(n,0) in native SQL queries on Oracle - Since Hibernate 6.0, columns of type number with scale 0 on Oracle were interpreted as **boolean**, **tinyint**, **smallint**, **int**, or **bigint**, depending on the precision. Now, columns of type number with scale 0 are interpreted as **int** or **bigint** depending on the precision.
- Removal of support for legacy database versions - Hibernate 6.2 introduces the concept of minimum supported database version for most of the database dialects that are supported by Hibernate.
- Changes to CDI handling - When CDI is available and configured, Hibernate can use the CDI **BeanManager** to resolve various bean references. Starting with Hibernate 6.2, these extensions will only be resolved from the CDI **BeanManager** if **hibernate.cdi.extensions** is set to **true**.
- Change in enhancement defaults and deprecation - The **enableLazyInitialization** and **enableDirtyTracking** enhancement tooling options, the global property **hibernate.bytecode.use_reflection_optimizer** as well as the respective **hibernate.enhancer.enableLazyInitialization** and **hibernate.enhancer.enableDirtyTracking** configuration settings, switched their default values to **true** and these settings are now deprecated.

- Package updates for **org.hibernate.cfg** and **org.hibernate.loader**: The **org.hibernate.cfg** and **org.hibernate.loader** packages have been updated to clearly display a distinction between contracts which are considered an API, SPI and internal.

- Changes in integration contracts (SPIs) - During the development of Hibernate ORM 6.2, the following SPIs have been modified: **EntityPersister#lock**, **EntityPersister#multiLoad**, **Executable#afterDeserialize**, and **JdbcType#getJdbcRecommendedJavaTypeMapping()**.

- Query Path comparison: As per Hibernate 6.2, comparisons of paths are type checked early.

- Batch Fetching and LockMode - When **LockMode** is greater than **READ**, Hibernate does not execute the batch fetching so existing uninitialized proxies will not be initialized. This is because the lock mode is different from one of the proxies in the batch fetch queue.

7.8.7. Migrating from Hibernate ORM 4.3 to Hibernate ORM 5.0

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-Jakarta Persistence transactions.
  - **jta**: Manage transactions using Jakarta Transactions.

IMPORTANT

If a Jakarta Persistence 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-Jakarta Persistence 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 Jakarta Transactions. A non-Jakarta Persistence application that uses Jakarta 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 Jakarta 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.

7.8.8. 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 performance improvements and bug fixes. For more information, see Hibernate ORM 5.1 Features in the JBoss EAP *Release notes for 7.1.0*. 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 focused on the following areas:

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

The schema management tooling changes are only a migration concern for applications that directly use 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 its delegates

Schema management tooling changes in JBoss EAP 7.1

Hibernate ORM 5.1.10, included in JBoss EAP 7.1, introduced a strategy for retrieving database tables that improve **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 of executing 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.

7.8.9. Migrating from Hibernate ORM 5.1 and Hibernate ORM 5.2 to Hibernate ORM 5.3

JBoss EAP 7.4 includes Hibernate ORM 5.3. This section highlights some of the changes needed when migrating from Hibernate ORM 5.1 to Hibernate ORM 5.2 and then 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 Jakarta Persistence 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 Hibernate ORM 5.2 Migration Guide.

Hibernate ORM 5.3 features

Hibernate ORM 5.3 adds support for the Jakarta Persistence 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.
  - Jakarta Persistence 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 Jakarta Persistence. 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 Jakarta Persistence 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 Jakarta Persistence 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.
- Jakarta Persistence 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.

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 Jakarta Persistence 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 Jakarta Persistence specification. This setting is ignored for a **SessionFactory** built using Jakarta Persistence bootstrapping.

Compatibility transformer

JBoss EAP 7.4 includes 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 7.4. This is a temporary solution and you should replace these method calls with the recommended Jakarta Persistence 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.hibernate.Session.getHibernateFlushMode()

Hibernate 5.1 Reference or Method	Transformed to Hibernate 5.3 Reference or Method
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)

7.9. HIBERNATE SEARCH CHANGES

The version of Hibernate Search that is included with JBoss EAP 7 has changed. The previous release of JBoss EAP included Hibernate Search 4.6.x. JBoss EAP 7 is included 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 5.5.8.Final 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.

In JBoss EAP 8.0, Hibernate Search 5 APIs have been removed and are replaced with Hibernate Search 6 APIs.

Additional resources

- Hibernate search changes

7.9.1. Hibernate Search 6 replaces Hibernate Search 5 APIs

Hibernate Search 5 APIs have been removed and are replaced with Hibernate Search 6 APIs in JBoss EAP 8.0.

To view a list of the removed features, see Hibernate Search 5 APIs Deprecated in JBoss EAP 7.4 and removed in EAP 8.0.

NOTE

Hibernate Search 6 APIs are **backwards-incompatible** with Hibernate Search 5 APIs. You will need to migrate your applications to Hibernate Search 6.

The latest version of Hibernate Search 6 included in JBoss EAP 8.0 is 6.2. If you are migrating from Hibernate Search 5, you should take into account the migration to version 6.0, 6.1, **and** 6.2.

See the following migrations guides for more information:

- To migrate your applications from Hibernate Search 5, see the Hibernate Search 6.0 migration guide.

- To migrate your applications from Hibernate Search 6.0 to 6.1, see the Hibernate Search 6.1 migration guide.
- To migrate your applications from Hibernate Search 6.1 to 6.2, see the Hibernate Search 6.2 migration guide

NOTE

Hibernate Search 6.2 is compatible with Hibernate ORM 6.2. For more information, see the section Hibernate ORM 6 in the Hibernate Search 6.2 Reference documentation.

7.9.2. Hibernate Search 6 supports Elasticsearch

JBoss EAP 8.0 also provides support for using an Elasticsearch backend in Hibernate Search 6 to index data into remote Elasticsearch or OpenSearch clusters.

To see a list of possible Hibernate Search architectures and backends, see Table 2. Comparison of architectures in the Hibernate Search 6.2 reference documentation.

For more information about configuring Hibernate Search 6, see Using Hibernate Search in the WildFly Developer guide.

Additional resources

- Hibernate changes

7.10. MIGRATE ENTITY BEANS TO JAKARTA PERSISTENCE

Support for *Enterprise Java Beans* entity beans is optional in Java EE 8 and they are not supported starting with 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**.

Starting with JBoss EAP 7, you must replace any CMP and BMP entity beans in your code with Jakarta Persistence entities. Jakarta Persistence entities are created using the jakarta.persistence.* classes and are defined in the **persistence.xml** file.

The following is an example of a Jakarta Persistence entity class:

```
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.Id;
import jakarta.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 xmlns="https://jakarta.ee/xml/ns/persistence"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="https://jakarta.ee/xml/ns/persistence
          https://jakarta.ee/xml/ns/persistence/persistence_3_0.xsd"
        version="3.0">
  <persistence-unit name="my-unique-persistence-unit-name">
    <properties>
      // properties...
    </properties>
  </persistence-unit>
</persistence>
```

For working examples of Jakarta Persistence entities, see the **cmt** quickstart that is included with JBoss EAP 8.0.

7.11. JAKARTA PERSISTENCE PROPERTY CHANGES

This section describes the Jakarta Persistence property changes introduced in JBoss EAP 7.0 and 7.1.

Jakarta 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-Jakarta Transactions 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 Jakarta Persistence 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.

Jakarta 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 Jakarta Transactions. 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

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 Jakarta Transactions as if they are synchronized persistence contexts.

7.12. MIGRATE JAKARTA ENTERPRISE BEANS CLIENT CODE

This section discusses the changes in Jakarta Enterprise Beans client in JBoss EAP 7.0. It also explains how to modify your client code to use the new default remote port and connector in JBoss EAP 7.0. In addition, it describes the required JBoss EJB client changes introduced in JBoss EAP 7.1 and JBoss EAP 7.0.

NOTE

Starting with JBoss EAP 7.0, enterprise entity beans are not supported. For more information, see Migrate Entity Beans to Jakarta Persistence .

7.12.1. Jakarta Enterprise Beans client changes in JBoss EAP 7

Starting with JBoss EAP 7, the default remote connector and port have been changed. For details about this change, see Update the Remote URL connector and port .

If you used the JBoss Server Migration Tool to migrate your server configuration, the old settings are preserved and you do not need to make the changes detailed here. However, if you use the new JBoss EAP 8.0 default configuration, you must make the following changes.

7.12.1.1. 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 8 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
```

7.12.1.2. Update the default connector

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

- If a client application uses the Jakarta Enterprise Beans 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 Jakarta Enterprise Beans 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
  ```

7.12.2. 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
```

7.12.3. Additional JBoss EJB client changes introduced in JBoss EAP 7.1

JBoss EAP 7.0 is included with JBoss Enterprise Java Beans client 2.1.4, JBoss EAP 7.1 and later was included with JBoss Enterprise Java Beans client 4.0.x, which includes a number of changes to the API.

NOTE

Starting with JBoss EAP 7, enterprise entity beans are not supported. For information about how to migrate entity beans to Jakarta Persistence, see Migrate Entity Beans to Jakarta Persistence.

- The **org.ejb.client.EJBClientInvocationContext** class adds the following new methods:

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 must be asynchronous regardless of whether the server-side method is asynchronous.
isIdempotent()	boolean	Determine whether the method is marked idempotent, meaning that the method method 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, is introduced for controlling access to privileged Enterprise Java Beans 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 is introduced for locating a specific Enterprise Java Beans 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.

Method	Type	Description
getParameterCount()	int	Get the parameter count.
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 is 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 is 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** is 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.
equals(Object other)	boolean	Indicates whether another object is equal to this one.

Method	Type	Description
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 deprecates the following methods:
  - **toAbstractEJBMetaData()**
  - **EJBMetaDataImpl(AbstractEJBMetaData<?,?>)**

7.13. MIGRATE CLIENTS TO USE THE WILDFLY CONFIGURATION FILE

Prior to release JBoss EAP 7.1, JBoss EAP client libraries, such as Enterprise Java Beans 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 an Enterprise Java Beans 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:1.7">
    <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:3.2">
        <connections>
            <connection uri="remote+http://127.0.0.1:8080" />
        </connections>
    </jboss-ejb-client>
</configuration>
```

Additional resources

- Configure Client Authentication with Elytron Client.
- Client Configuration Using the **wildfly-config.xml** File.

7.14. 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 archive 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.

7.15. 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.

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. Since 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 7.4 *Development Guide*.

Migrate Custom Authenticator Valves

For information about how to migrate authenticator valves, see Migrate authenticator valves.

7.16. SECURITY APPLICATION CHANGES

The replacement of JBoss Web with Undertow requires changes to security configuration since JBoss EAP 7. Starting with JBoss EAP 8.0, you must use Elytron as the legacy security since PicketBox is no longer available.

7.16.1. 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 JBoss EAP 7.4 *Development Guide*.

7.16.2. PicketLink removal

PicketLink has been removed from JBoss EAP 8.0.

PicketLink SP

Use Keycloak SAML adapter instead of the PicketLink service provider.

To migrate from PicketLink by configuring the Keycloak SAML adapter, perform the following tasks:

- Install Keycloak SAML client to JBoss EAP 8.0. For more information, see
  - Installing JBoss EAP 8.0 using the jboss-eap-installation-manager
  - Keycloak SAML adapter feature pack for securing applications using SAML
- Configure a Keycloak SAML instead of PicketLink IdP if needed. To secure the SP application using Keycloak SAML, a SAML client needs to be created. For more information on creating an Keycloak SAML client, see Creating a SAML client in the JBoss EAP 7.5 *Server Administration Guide*.
- Update the applications to use the Keycloak SAML adapter. For more information on updating the applications, see Securing web applications using SAML.

PicketLink IDP

PicketLink IDP is not available since JBoss EAP 8.0 and you can configure Red Hat build of Keycloak instead. For more information, see Configuring Red Hat build of Keycloak.

PicketLink STS

In previous releases, you could configure PicketLink STS as an alternative to the Apache CXF Security Token Service implementation. PicketLink STS configuration involved a legacy security domain. Any references to legacy security domains and PicketLink in the STS application needs to be removed, so you must configure Apache CXF STS instead.

For more information on how to configure Apache CXF STS, see Security Token Service (STS) in the JBoss EAP 7.4 *Developing Web Services Applications* .

7.16.3. Vault removal

Vaults has been removed from JBoss EAP 8.0. If your applications use legacy vault expressions, you must migrate and use Elytron encrypted expressions.

Check for instances of **${VAULT::** in your deployment files, which could be in annotations or deployment descriptors, and replace them with the corresponding encrypted expressions.

Additional resources

- Encrypted expressions in Elytron

7.16.4. OIDC client migration

The Keycloak OIDC client adapter is not supported in JBoss EAP 8.0 and is replaced by the native Elytron OIDC client, providing similar functionality and configuration.

To migrate from the Keycloak OIDC client adapter to the native Elytron OIDC client, follow these steps:

- Check for **<auth-method>KEYCLOAK</auth-method>** in the **web.xml** file of the application and replace it with **<auth-method>OIDC</auth-method>** in the **web.xml** file of the deployment.
- Check for the presence of **WEB-INF/keycloak.json** and rename it to **WEB-INF/oidc.json**.

Additional resources

- OpenID Connect configuration in JBoss EAP
- Migrate **keycloak** subsystem

7.16.5. Custom login modules migration

In JBoss EAP 8.0, the legacy security subsystem has been removed. To continue using your custom login modules with the **elytron** subsystem, use the new Java Authentication and Authorization Service (JAAS) security realm and **jaas-realm**.

Additional resources

- JAAS realm in the **elytron** subsystem

7.16.6. Other security application changes

There are a few noticeable differences between JBoss EAP 7.2 or higher and earlier versions:

- The **NegotiationAuthenticator** valve is not required in the **jboss-web.xml**, but there still must be **<security-constraint>** and **<login-config>** elements defined in the **web.xml**. These are used to decide which resources are secured.
- The **auth-method** element in the **<login-config>** element is now a comma-separated list. The exact value **SPNEGO** must be there and should appear first in that list. In cases where **FORM** authentication is desired as a fallback, the exact value would be **SPNEGO,FORM**.

- The **jboss-deployment-structure.xml** file is not required.

7.17. JBOSS LOGGING CHANGES

Starting with JBoss EAP 7, if your application uses JBoss Logging, be aware that the annotations in the **org.jboss.logging** package are deprecated. 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 7.1. 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

7.18. JAKARTA FACES CODE CHANGES

This section describes the impact of the Jakarta Faces code changes in migrating your application to JBoss EAP.

Dropped support for Jakarta Server Faces prior to 4.0

NOTE

Jakarta Server Faces is the new name for JavaServer Faces.

With JBoss EAP 6.4, you could continue to use Jakarta Server Faces 1.2 with your application deployment by creating a **jboss-deployment-structure.xml** file. JBoss EAP 7.4 includes Jakarta Server Faces 2.3 and no longer supports the Jakarta Server Faces 1.2 API. If your application uses Jakarta Server Faces 1.2, you must rewrite it to use Jakarta Server Faces 2.3.

JBoss EAP 8.0 no longer supports any version of JSF prior to 4.0.

7.19. INTEGRATE MYFACES FOR ALTERNATIVE FACES

You can simplify the installation of an alternative Jakarta Faces implementation, **MyFaces**, as an alternative to the default Mojarra Jakarta Faces implementation within the JBoss EAP by introducing the Galleon feature pack, **eap-myfaces-feature-pack**. You can use this feature pack to provision a different Jakarta Faces implementation within JBoss EAP.

You can use the **eap-myfaces-feature-pack** to select the **MyFaces** version by using the **MYFACES_VERSION** environment variable. This feature pack introduces a single layer named **MyFaces**, providing the option to install and select **MyFaces** as an alternative. For more information, see How to configure the Multi-JSF feature in EAP 8 .

NOTE

Compatibility with JBoss EAP 8.0 is limited to versions 4.x and above.

7.20. 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 7.4 *Development Guide*.

7.21. APPLICATION CLUSTERING CHANGES

This section provides an overview of the clustering changes required for migrating your application from JBoss EAP 6 to JBoss EAP 8. In addition, this section describes how clustering changes might impact the migration of your applications to JBoss EAP 8.0.

7.21.1. 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 8.0.

- 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 7.4 *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 7.4 *Development Guide*.

- You can now define clustered singleton MDBs. For more information, see Clustered Singleton MDBs in the JBoss EAP 7.4 *Developing Jakarta Enterprise Beans Applications* .

- JBoss EAP 8.0 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 7.4 *Configuration Guide*.

7.21.2. 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 **distributable-web** subsystem deprecates the **<replication-config>** element of **jboss-web.xml**. It enhances the usage of **<replication-config>** by generating an ad hoc distributable web session profile.

You can override the default distributable session management behavior by referencing a session management profile by name or by providing a deployment-specific session management configuration. For more information, see Overriding the default distributable session management behavior .

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/>**. 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/>	Starting with JBoss EAP 7, this configuration element and its sub-elements are no longer used.
<use-session-passivation/>	Previously, passivation was enabled using this attribute. 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. 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. 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/>**.

Configuration Element	Description of Change
<replication-config/>	The **distributable-web** subsystem deprecates this element. For more information, see The distributable-web subsystem for Distributable Web Session Configurations in the JBoss EAP 7.4 *Development Guide* and Overriding the default distributable session management behavior.
<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 7.4 *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/>**. 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. 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.
<session-notification-policy/>	Previously, the value specified by this attribute defined a policy for triggering session events. 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.

7.21.3. Overriding the default distributable session management behavior

You can override the default distributable session management behavior in one of the following ways:

- Referencing a session management profile by name
- Providing a deployment-specific session management configuration

Referencing an existing session management profile

- To use an existing distributed session management profile, include a **distributable-web.xml** deployment descriptor located in the application's /**WEB-INF** directory. For example:

/WEB-INF/distributable-web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<distributable-web xmlns="urn:jboss:distributable-web:1.0">
    <session-management name="foo"/>
</distributable-web>
```

- Alternatively, define the target distributed session management profile within an existing **jboss-all.xml** deployment descriptor:

/META-INF/jboss-all.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss xmlns="urn:jboss:1.0">
    <distributable-web xmlns="urn:jboss:distributable-web:1.0">
        <session-management name="foo"/>
    </distributable-web>
</jboss>
```

Using a Deployment-specific Session Management Profile
If only a single web application uses the custom session management configuration, you can define the configuration within the deployment descriptor itself. Ad hoc configuration looks identical to the configuration used by the **distributable-web** subsystem.

- Define the custom session management configuration within the deployment descriptor. For example:

/WEB-INF/distributable-web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<distributable-web xmlns="urn:jboss:distributable-web:1.0">
    <infinispan-session-management cache-container="foo" cache="bar" granularity="SESSION">
        <primary-owner-affinity/>
    </infinispan-session-management>
</distributable-web>
```

- Alternatively, define the session management configuration within an existing **jboss-all.xml** deployment descriptor:

/META-INF/jboss-all.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss xmlns="urn:jboss:1.0">
    <distributable-web xmlns="urn:jboss:distributable-web:1.0">
        <infinispan-session-management cache-container="foo" cache="bar" granularity="ATTRIBUTE">
            <local-affinity/>
        </infinispan-session-management>
    </distributable-web>
</jboss>
```

7.21.4. 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>
```

Starting with JBoss EAP 7, you no longer need 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 Enterprise Java Beans 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.

7.21.5. 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 7.4 *Development Guide*.

7.21.6. 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 8.

For more information about HA singleton services, see HA Singleton Service in the JBoss EAP 7.4 *Development Guide*. For information about HA singleton deployments, see HA Singleton Deployments in the JBoss EAP 7.4 *Development Guide*.

7.22. CONTEXTSERVICE CUSTOMIZATION BY USING CONTEXT TYPES

As part of Jakarta EE Concurrency 3.0, you can customize the **ContextService** property by using context types. The Transaction context is one such type. The Transaction context replaces the use of the **use-transaction-setup-provider** resource-definition attribute. When the **use-transaction-setup-provider** attribute is set to **true**, the transaction context is cleared and when this attribute is set to **false**, the transaction context is unchanged.

Red Hat no longer supports vendor-specific configurations and therefore, such resource-definition attributes have been deprecated. In JBoss EAP 7, the default configurations defined the **use-transaction-setup-provider** attribute as **false**, which means that the transaction context was unchanged when a contextual task was run on a thread. By default, in JBoss EAP 8, the default **ContextService** property is aligned with the Jakarta EE Concurrency 3.0 specification, and clears the transaction context before a contextual task is executed.

To use a different **ContextService**, you must define it on the deployment by using the **ContextServiceDefinition** annotation or by specifying it in XML.

7.23. REMOVAL OF DEPRECATED INITIALCONTEXT CLASS

The **org.jboss.naming.remote.client.InitialContextFactory** class is removed in JBoss EAP 8. In JBoss EAP 7, the **org.jboss.naming.remote.client.InitialContextFactory** class had been deprecated and replaced with the **org.wildfly.naming.client.WildFlyInitialContextFactory** class. You must migrate your source code or configuration files to reflect this change.

Naming configuration changes:

- If a user application is using system or environment properties, then **java.naming.factory.initial** property must be migrated from **java.naming.factory.initial=org.jboss.naming.remote.client.InitialContextFactory** to **java.naming.factory.initial=org.wildfly.naming.client.WildFlyInitialContextFactory**.

- If a user application is using WSDL contracts that contain **<soapjms:jndiInitialContextFactory>**, then their values must be migrated from **<soapjms:jndiInitialContextFactory>org.jboss.naming.remote.client.InitialContextFactory**

<soapjms:jndiInitialContextFactory> to **<soapjms:jndiInitialContextFactory>org.wildfly.naming.client.WildFlyInitialContextFactory<soapjms:jndiInitialContextFactory>**.

- If a user application is using Java code to configure remote naming, then it must be updated from **Properties env = new Properties();env.put(Context.INITIAL_CONTEXT_FACTORY, org.jboss.naming.remote.client.InitialContextFactory.class.getName());** to **env.put(Context.INITIAL_CONTEXT_FACTORY, org.wildfly.naming.client.WildFlyInitialContextFactory.class.getName());**.

The methods listed below, from the **org.wildfly.naming.client.ProviderEnvironment** class, have been deprecated in JBoss EAP 7 and are now removed in JBoss EAP 8 as part of Red Hat's commitment to replacing problematic language in our code, documentation, and web properties. For more details, see our CTO Chris Wright's message .

Any code containing a removed method must be refactored using the corresponding replacement:

- **getBlackList()** replaced by **getBlockList()**
- **updateBlacklist(URI)** replaced by **updateBlockList(URI)**
- **dropFromBlacklist(URI)** replaced by **dropFromBlocklist(URI)**

7.24. RESOURCE ADAPTERS

A Jakarta Connectors Resource Adapter lets your applications communicate with any messaging provider. It configures how Jakarta EE components such as MDBs and other Jakarta Enterprise Beans, and even Servlets, can send or receive messages.

7.24.1. Deploying the IBM MQ Resource Adapter

IBM MQ is the Messaging Oriented Middleware (MOM) product offering from IBM that allows applications on distributed systems to communicate with each other. This is accomplished through the use of messages and message queues. IBM MQ is responsible for delivering messages to the message queues and for transferring data to other queue managers using message channels. For more information about IBM MQ, see IBM MQ on the IBM products website.

Summary

IBM MQ can be configured as an external Java Message Service provider for JBoss EAP 8.0. This section covers the steps to deploy and configure the IBM MQ resource adapter in JBoss EAP. This deployment and configuration can be accomplished by using the management CLI tool or the web-based management console. See JBoss EAP supported configurations for the most current information about the supported configurations of IBM MQ.

NOTE

You must restart your system after configuring your IBM MQ resource adapter for the configuration changes to take effect.

JBoss EAP 8.0 is a Jakarta EE 10 implementation, so the packages used for all EE APIs have changed from javax to jakarta, which requires a Jakarta EE 10 compliant resource adapter. If you were using the IBM MQ Resource adapter in JBoss EAP 7.x or earlier, you must use **wmq.jakarta.jmsra.rar**, the IBM MQ Resource Adapter that uses this jakarta namespace.

- Remove and undeploy the previous resource adapter configuration for **wmq.jmsra.rar** and use **wmq.jakarta.jmsra.rar**
- Deploy **wmq.jakarta.jmsra.rar** and configure as per the steps provided in this section.

Prerequisites

Before you get started, you must verify the version of the IBM MQ resource adapter and understand its configuration properties.

- The IBM MQ resource adapter is supplied as a Resource Archive (RAR) file called **wmq.jakarta.jmsra.rar**. You can obtain the **wmq.jakarta.jmsra.rar** file from **/opt/mqm/java/lib/jca/wmq.jakarta.jmsra.rar**. See JBoss EAP supported configurations for information about the specific versions that are supported for each release of JBoss EAP.
- You must know the following IBM MQ configuration values. Refer to the IBM MQ product documentation for details about these values.
    - *MQ_QUEUE_MANAGER*: The name of the IBM MQ queue manager
    - *MQ_HOST_NAME*: The host name used to connect to the IBM MQ queue manager
    - *MQ_CHANNEL_NAME*: The server channel used to connect to the IBM MQ queue manager
    - *MQ_QUEUE_NAME*: The name of the destination queue
    - *MQ_TOPIC_NAME*: The name of the destination topic
    - *MQ_PORT*: The port used to connect to the IBM MQ queue manager
    - *MQ_CLIENT*: The transport type
- For outbound connections, you must also be familiar with the following configuration value:
    - *MQ_CONNECTIONFACTORY_NAME*: The name of the connection factory instance that will provide the connection to the remote system

Procedure

The following are default configurations provided by IBM and are subject to change. Refer to the IBM MQ documentation for more information.

1. First, deploy the resource adapter manually by copying the **wmq.jakarta.jmsra.rar** file to the ***EAP_HOME*/standalone/deployments/** directory.
2. Next, use the management CLI to add the resource adapter and configure it.

   ```
   /subsystem=resource-adapters/resource-
   adapter=wmq.jakarta.jmsra.rar:add(archive=wmq.jakarta.jmsra.rar, transaction-
   support=XATransaction)
   ```

   Note that the **transaction-support** element was set to **XATransaction**. When using transactions, be sure to supply the security domain of the XA recovery datasource, as in the example below.

   ```
   /subsystem=resource-adapters/resource-adapter=test/connection-definitions=test:write-
   attribute(name=recovery-security-domain,value=myDomain)
   ```

For more information about XA Recovery, see Configuring XA Recovery in the JBoss EAP 7.4 *Configuration Guide*.

For non-transactional deployments, change the value of **transaction-support** to **NoTransaction**.

```
/subsystem=resource-adapters/resource-
adapter=wmq.jakarta.jmsra.rar:add(archive=wmq.jakarta.jmsra.rar, transaction-
support=NoTransaction)
```

3. Now that the resource adapter is created, you can add the necessary configuration elements to it.

   a. Add an **admin-object** for queues and configure its properties.

```
/subsystem=resource-adapters/resource-adapter=wmq.jakarta.jmsra.rar/admin-
objects=queue-ao:add(class-
name=com.ibm.mq.jakarta.connector.outbound.MQQueueProxy, jndi-
name=java:jboss/MQ_QUEUE_NAME)

/subsystem=resource-adapters/resource-adapter=wmq.jakarta.jmsra.rar/admin-
objects=queue-ao/config-properties=baseQueueName:add(value=MQ_QUEUE_NAME)

/subsystem=resource-adapters/resource-adapter=wmq.jakarta.jmsra.rar/admin-
objects=queue-ao/config-
properties=baseQueueManagerName:add(value=MQ_QUEUE_MANAGER)
```

   b. Add an **admin-object** for topics and configure its properties.

```
/subsystem=resource-adapters/resource-adapter=wmq.jakarta.jmsra.rar/admin-
objects=topic-ao:add(class-
name=com.ibm.mq.jakarta.connector.outbound.MQTopicProxy, jndi-
name=java:jboss/MQ_TOPIC_NAME)

/subsystem=resource-adapters/resource-adapter=wmq.jakarta.jmsra.rar/admin-
objects=topic-ao/config-properties=baseTopicName:add(value=MQ_TOPIC_NAME)

/subsystem=resource-adapters/resource-adapter=wmq.jakarta.jmsra.rar/admin-
objects=topic-ao/config-
properties=brokerPubQueueManager:add(value=MQ_QUEUE_MANAGER)
```

   c. Add a connection definition for a managed connection factory and configure its properties.

```
/subsystem=resource-adapters/resource-adapter=wmq.jakarta.jmsra.rar/connection-
definitions=mq-cd:add(class-
name=com.ibm.mq.jakarta.connector.outbound.ManagedConnectionFactoryImpl, jndi-
name=java:jboss/MQ_CONNECTIONFACTORY_NAME, tracking=false)

/subsystem=resource-adapters/resource-adapter=wmq.jakarta.jmsra.rar/connection-
definitions=mq-cd/config-properties=hostName:add(value=MQ_HOST_NAME)

/subsystem=resource-adapters/resource-adapter=wmq.jakarta.jmsra.rar/connection-
definitions=mq-cd/config-properties=port:add(value=MQ_PORT)

/subsystem=resource-adapters/resource-adapter=wmq.jakarta.jmsra.rar/connection-
```

```
definitions=mq-cd/config-properties=channel:add(value=MQ_CHANNEL_NAME)

/subsystem=resource-adapters/resource-adapter=wmq.jakarta.jmsra.rar/connection-
definitions=mq-cd/config-properties=transportType:add(value=MQ_CLIENT)

/subsystem=resource-adapters/resource-adapter=wmq.jakarta.jmsra.rar/connection-
definitions=mq-cd/config-
properties=queueManager:add(value=MQ_QUEUE_MANAGER)
```

4. If you want to change the default provider for the EJB3 messaging system in JBoss EAP from JBoss EAP 8.0 messaging to IBM MQ, use the management CLI to modify the **ejb3** subsystem as follows:

```
/subsystem=ejb3:write-attribute(name=default-resource-adapter-
name,value=wmq.jakarta.jmsra.rar)
```

5. Configure the **@ActivationConfigProperty** and **@ResourceAdapter** annotations in the MDB code as follows:

```
@MessageDriven(name="IbmMqMdb", activationConfig = {
   @ActivationConfigProperty(propertyName = "destinationType",propertyValue =
"jakarta.jms.Queue"),
   @ActivationConfigProperty(propertyName = "useJNDI", propertyValue = "false"),
   @ActivationConfigProperty(propertyName = "hostName", propertyValue =
"MQ_HOST_NAME"),
   @ActivationConfigProperty(propertyName = "port", propertyValue = "MQ_PORT"),
   @ActivationConfigProperty(propertyName = "channel", propertyValue =
"MQ_CHANNEL_NAME"),
   @ActivationConfigProperty(propertyName = "queueManager", propertyValue =
"MQ_QUEUE_MANAGER"),
   @ActivationConfigProperty(propertyName = "destination", propertyValue =
"MQ_QUEUE_NAME"),
   @ActivationConfigProperty(propertyName = "transportType", propertyValue =
"MQ_CLIENT")
})

@ResourceAdapter(value = "wmq.jakarta.jmsra-VERSION.rar")
@TransactionAttribute(TransactionAttributeType.NOT_SUPPORTED)
public class IbmMqMdb implements MessageListener {
}
```

Be sure to replace the *VERSION* in the **@ResourceAdapter** value with the actual version in the name of the RAR.

6. Activate your resource adapter:

```
/subsystem=resource-adapters/resource-adapter=wmq.jakarta.jmsra.rar:activate()
```

7.24.1.1. Limitations and known issues of IBM MQ resource adapters

The following table lists known issues with the IBM MQ resource adapters. A checkmark (✔) in the version column indicates the issue is a problem for that version of the resource adapter.

Table 7.2. Known Issues with the IBM MQ Resource Adapters

JIRA	Description of Issue	IBM MQ 9
JBEAP-503	The IBM MQ resource adapter returns different String values for the **Queue.toString()** and **QueueBrowser.getQueue().toString()** methods. **Queue** is instance of the **com.ibm.mq.connector.outbound.MQQueueProxy** class, which is different from the **com.ibm.mq.jms.MQQueue** class that is returned by the **QueueBrowser.htmlQueueBrowser.getQueue()** method. These classes contain different implementations of the **toString()** method. Be aware that you cannot rely on these **toString()** methods to return the same value.	✔
JBEAP-511, JBEAP-550, JBEAP-3686	The following restrictions apply to message property names for IBM MQ. ● In the **activation-config** section of the deployment descriptor, you must not configure the **destinationName** property using special characters such as _, **&**, or **|**. Use of these characters causes the MDB deployment to fail with a **com.ibm.msg.client.jms.DetailedInvalidDestinationException** exception. ● In the **activation-config** section of the deployment descriptor, you must not configure the **destinationName** property using the **java:/** prefix. Use of this prefix causes the MDB deployment to fail with a **com.ibm.msg.client.jms.DetailedInvalidDestinationException** exception. ● A property must not begin with "JMS" or "usr.JMS" as they are reserved for use by IBM MQ JMS classes. Exceptions are noted on the IBM Knowledge Center website. See Property name restrictions for IBM MQ, Version 9.0on the IBM Knowledge Center website for the complete list of message property name restrictions.	✔
JBEAP-549	When specifying the **destination** property name value for an MDB using the **@ActivationConfigProperty** annotation, you must use all upper case letters. For example: `@ActivationConfigProperty(propertyName = "destination", propertyValue = "QUEUE")`	✔

JIRA	Description of Issue	IBM MQ 9

JBEAP-624	If the IBM MQ resource adapter is used to create a connection factory in a Jakarta EE deployment using the **@JMSConnectionFactoryDefinition** annotation, you must specify the **resourceAdapter** property. Otherwise, the deployment will fail.
	
	`@JMSConnectionFactoryDefinition(`
	`    name = "java:/jms/WMQConnectionFactory",`
	`    interfaceName = "javax.jms.ConnectionFactory",`
	`    resourceAdapter = "wmq.jmsra",`
	`    properties = {`
	`        "channel=<channel>",`
	`        "hostName=<hostname_wmq_broker>",`
	`        "transportType=<transport_type>",`
	`        "queueManager=<queue_manager>"`
	`    }`
	`)`
		✔

JBEAP-2339	The IBM MQ resource adapter is able to read messages from queues and topics even before the connection has started. This means a consumer can consume messages before the connection is started. To avoid hitting this issue, use connection factories created by the remote IBM MQ broker using the **external-context** and not connection factories created by IBM MQ resource adapter.	✔

JBEAP-3685	Once **<transaction-support>XATransaction</transaction-support>** is set, a **JMSContext** is always **JMSContext.SESSION_TRANSACTED**, whether it was created using injection or manually.
	
	In the following code example, the **@JMSSessionMode(JMSContext.DUPS_OK_ACKNOWLEDGE)** is ignored and the **JMSContext** remains at **JMSContext.SESSION_TRANSACTED**.
	
	`@Inject`
	`@JMSConnectionFactory("jms/CF")`
	`@JMSPasswordCredential(userName="myusername",`
	`password="mypassword")`
	`@JMSSessionMode(JMSContext.DUPS_OK_ACKNOWLEDGE)`
	`transient JMSContext context3;`
		✔

JBEAP-14633	According to the JMS specification, the **QueueSession** interface cannot be used to create objects specific to the publish/subscribe domain and certain methods that inherit from **Session** should throw an **javax.jms.IllegalStateException**. One such method is such **QueueSession.createTemporaryTopic()**. Instead of throwing an **javax.jms.IllegalStateException**, the IBM MQ resource adapter throws a **java.lang.NullPointerException**.	✔

JIRA	Description of Issue	IBM MQ 9
JBEAP-14634	The **MQTopicProxy.getTopicName()** returns different topic name than was set by the IBM MQ broker. For example, if the topic name was set to **topic://MYTOPIC?XMSC_WMQ_BROKER_PUBQ_QMGR=QM**, the **MQTopicProxy** returns **topic://MYTOPIC**.	✔
JBEAP-14636	The default **autoStart** setting for the **JMSContext** is **false**, meaning the underlying connection used by the **JMSContext** is not started automatically when a consumer is created. This setting should default to **true**.	✔

JBEAP-14640	The IBM MQ resource adapter throws **DetailedJMSException** instead of a **JMSSecurityException** when invalid credentials are used and logs the following error to the server console.
	
	`WARN [org.jboss.jca.core.connectionmanager.pool.strategy.PoolByCri] (EJB default - 7) IJ000604: Throwable while attempting to get a new connection: null: com.ibm.mq.connector.DetailedResourceException: MQJCA1011: Failed to allocate a JMS connection., error code: MQJCA1011 An internal error caused an attempt to allocate a connection to fail. See the linked exception for details of the failure.`
	
	The following is an example of code that can cause this issue.
	
	`QueueConnection qc = queueConnectionFactory.createQueueConnection("invalidUserName", "invalidPassword");`
		✔

JBEAP-14642	Due to an invalid class cast conversion by the resource adapter in the **MQMessageProducer.send(Destination destination, Message message)** and **MQMessageProducer.send(Destination destination, Message message, int deliveryMode, int priority, long timeToLive, CompletionListener completionListener)** methods, the IBM MQ resource adapter throws a **JMSException** and logs the following error message to the server console. `SVR-ERROR: Expected JMSException, received com.ibm.mq.connector.outbound.MQQueueProxy cannot be cast to com.ibm.mq.jms.MQDestination` This is because the JNDI name used in the queue or topic lookup is **com.ibm.mq.connector.outbound.MQQueueProxy/MQTopicProxy**.	✔
JBEAP-14643	The **setDeliveryDelay(expDeliveryDelay)** method on the **JMSProducer** interface does not change the setting. After calling this method, it remains at the default setting of **0**.	✔

JIRA	Description of Issue	IBM MQ 9
JBEAP-14670	If work is done on a **QueueSession** that was created prior to a **UserTransaction.begin()**, that work is not considered part of the transaction. This means that any message sent to the queue using this session is not committed by a **UserTransaction.commit()**, and after a **UserTransaction.rollback()**, the message remains on the queue.	✔

JBEAP-14675	If you close a connection and then immediately create a **JMSContext** with the same **clientID**, the IBM MQ resource adapter intermittently logs the following error to the server console.
	
	`ERROR [io.undertow.request] (default task-1) UT005023: Exception handling request to /jmsServlet-1.0-SNAPSHOT/: com.ibm.msg.client.jms.DetailedJMSRuntimeException: MQJCA0002: An exception occurred in the IBM MQ layer. See the linked exception for details.`
	`A call to IBM MQ classes for Java(tm) caused an exception to be thrown.`
	
	This issue does not occur when there is a delay in creating the new **JMSContext** after the connection with the same**clientID** is closed.
		✔

JBEAP-15535	If a stateful session bean tries to send a message to a topic while in a container managed transaction (CMT), the message send fails with the following message.
	
	`SVR-ERROR: com.ibm.msg.client.jms.DetailedJMSException: JMSWMQ2007: Failed to send a message to destination '`*MDB_NAME TOPIC_NAME*`'`
	
	The stack trace shows it to be caused by the following exception.
	
	`com.ibm.mq.MQException: JMSCMQ0001: IBM MQ call failed with compcode '2' ('MQCC_FAILED') reason '2072' ('MQRC_SYNCPOINT_NOT_AVAILABLE')`	

JBEAP-25561	When a message is sent to the destination with the **JMSReplyTo** header set, it is modified after reaching the IBM MQ 9 broker. As a result, the reply message of the response is directed to the destination defined in the modified **JMS_REPLY** header. For example, if the **JMSReplyTo** header is set to queue with the name **queue:///MYQUEUE** using the header **message.setJMSReplyTo(queue);** and sent to a IBM MQ 9.3 broker with queue manager named **QM**, then its name is changed to **queue://QM/MYQUEUE**.	✔

7.24.2. Removal of Apache Log4j version 1 APIs

Starting with JBoss EAP 8, support for Apache **Log4j** version 1 APIs has been stopped. Any application that is not packaging **log4j.jar** and **log4j** configuration must be updated.

Impact:

Log messages will no longer be routed based on the logging subsystem. If an application is not packaging **log4j.jar** and any of the following statements are true, then migration changes are required:

- If you use **log4j** in your deployment and do not include a **log4j** configuration file, then you must either migrate to a new logging facade or add a **log4j** configuration to your deployment.
- If you use a **log4j.xml**, **log4j.properties**, or **jboss-log4j.xml** file in your deployment and are not packaging the **log4j.jar** in your application. If it is a **jboss-log4j.xml** file, then you must rename the file to **log4j.xml**.
- If you use **log4j** v1 appenders in the JBoss EAP Logging subsystem in a custom-handler, it will be no longer supported.
- If an application classes import classes such as **org.apache.log4j.Logger**.
- If the application includes a **jboss-deployment-structure.xml** or has **Dependencies:** specified in the **MANIFEST.MF** declaring a module dependency on **org.jboss.log4j.logmanager**, these dependencies will need to be removed.

Migration:

- Update the application classes to use Apache **Log4jv2** classes or use one of the other Logging APIs provided by JBoss EAP 8.
- Change the **org.apache.log4j.Logger (log4j v1)** class to **org.apache.logging.log4j.Logger (log4j v2)**.
- If an application packages **log4j.properties**, **log4j.xml**, or **jboss-log4j.xml**, you must::
  - Configure the logging in the JBoss EAP configuration.
  - Configure **logging.properties** in an application as the **log4jv2** configuration files are not supported in an application.

OR

- Package the Apache **Log4j** version 1 JAR in the application instead of depending on JBoss EAP 8 for the Logging APIs. You can also exclude the JBoss Logging APIs from the application by the **jboss-deployment-structure.xml exclude-subsystems** on the logging subsystem.

Additional Details:

Disabling implicit logging dependencies for a specific deployment

- In an application's **jboss-deployment-structure.xml**, configure **exclude-subsystems** to exclude the logging subsystem such as:

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.2">
 <deployment>
   <exclude-subsystems>
     <subsystem name="logging"/>
```

```
    </exclude-subsystems>
  </deployment>
</jboss-deployment-structure>
```

- If the application is an EAR file and has a sub-deployment named **example.war**, the **jboss-deployment-structure.xml** file is located in the EAR file location / **META-INF/jboss-deployment-structure.xml** and the logging subsystem will be excluded by declaring it in the sub-deployment such as:

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.2">
  <sub-deployment name="example.war">
    <exclude-subsystems>
      <subsystem name="logging"/>
    </exclude-subsystems>
  </sub-deployment>
</jboss-deployment-structure>
```

Disabling implicit logging dependencies for all deployments

To make logging APIs unavailable to deployments by default, set **add-logging-api-dependencies** to **false** by using the following CLI command:

```
/subsystem=logging:write-attribute(name="add-logging-api-dependencies", value="false")
```

To set the JBoss Module and Logging API as a dependency, modify the **jboss-deployment-structure.xml** or **MANIFEST.MF** configuration files:

```
<subsystem xmlns="urn:jboss:domain:logging:8.0">
    <add-logging-api-dependencies value="false"/>
  ...
</subsystem>
```

NOTE

If an application packages Apache **Log4j v1** JARs and **log4j** configuration in an application: * Application logging is no longer managed by EAP, it is application managed. * An application should not attempt to write to the **server.log** as unexpected results can occur because logging frameworks are expected to be writing to specific log files.

For more information, see Apache Log4j version 1 is no longer provided in JBoss EAP 8.0 .

CHAPTER 8. MISCELLANEOUS CHANGES

This section provides an overview of the various miscellaneous changes happening in this release.

8.1. CHANGES TO DELIVERY OF JBOSS EAP NATIVES AND APACHE HTTP SERVER

JBoss EAP 8.0 natives are delivered differently in this release than in JBoss EAP 6. Some components include 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 dedicated 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 8.0
AIO Natives for Messaging	Delivered with the product in a separate "Native Utilities" download	Included within the JBoss EAP distribution.
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	Support for tcnatives was removed in JBoss EAP 7

Additional changes for JBoss EAP Natives and Apache HTTP Server

- 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 8.0 servers, you can do one of the following:
    - Use the Apache HTTP Server provided by JBoss Core Services.
    - You can configure JBoss EAP 8.0 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 7.4 *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 7.4 *Configuration Guide*.

- You can find more information about JBoss Core Services in the *Apache HTTP Server Installation Guide*.
    - You can configure JBoss EAP 8.0 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.
- You can find more information about JBoss Core Services in the *Apache HTTP Server Installation Guide*.

8.2. CHANGES TO DEPLOYMENTS ON AMAZON EC2

Several 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 start non-clustered and clustered JBoss EAP instances and domains in Amazon EC2 has changed significantly.
- In JBoss EAP 6, configuration depended on the **User Data:** field. In JBoss EAP 7, the AMI scripts that parsed the configuration in the **User Data:** field and started the servers automatically on instance startup have been removed.
- Red Hat JBoss Operations Network agent was installed in the JBoss EAP 6. Starting with JBoss EAP 7.0, you must install it separately.

For details on deploying JBoss EAP 7 on Amazon EC2, see *Deploying JBoss EAP on Amazon Web Services*.

8.3. REMOVE APPLICATIONS THAT INCLUDE SHARED MODULES

Changes introduced in the JBoss EAP 7.1 server and the Maven plug-in can result in the following failure when you attempt to remove 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 8.0.

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 updated undeploy behavior is more accurate and ensures that you do not end up in an unstable deployment state.

8.4. CHANGES TO THE ADD-USER SCRIPT

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. Starting with 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 7.4 *Configuration Guide*.

8.5. 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. Starting with JBoss EAP 7, this configuration is no longer supported and must be removed.

8.6. CHANGES IN SOAP WITH ATTACHMENTS API FOR JAVA

Update the user-defined SOAP handlers to comply with the SAAJ 3.0 specification when migrating to JBoss EAP 8.0.

Additional resources

- Jakarta Soap with Attachments

CHAPTER 9. MIGRATING TO 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.

Starting with JBoss EAP 8.0, you must use Elytron as the legacy security subsystem is not available for migrating your application.

9.1. OVERVIEW OF ELYTRON

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.

For an overview of the new resources that are available in the **elytron** subsystem, see Resources in the Elytron Subsystem in the JBoss EAP 7.4 *Security Architecture*.

9.2. MIGRATE SECURE VAULTS AND PROPERTIES

To use Elytron, you must migrate secure vaults to secure credential storage and migrate legacy security properties to Elytron security properties.

9.2.1. Migrate Secure Vaults to Secure Credential Storage

The secure vault 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. JBoss EAP 7.1 or later uses a credential store to store strings. Credential stores 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, you must replace the data with Elytron security properties. For more information about Elytron security properties, see Elytron security properties.

For more information about credential stores, see Credential Stores in the JBoss EAP 7.4 *How to Configure Server Security*.

9.2.1.1. Migrate 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. To execute the **vault** command, run the **elytron-tool** script in the ***EAP_HOME*/bin** directory.

```
$ EAP_HOME/bin/elytron-tool.sh 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
```

IMPORTANT

Credential stores are only used for securing passwords. The vault expression feature used in the management model is not supported. For more information, see Creating an encrypted expression in Elytron

Choose one of the following migration options:

- Migrating an Individual Security Vault to a Credential Store
- Migrating multiple security vaults to a credential store in bulk

NOTE

- The WildFly Elytron Tool cannot handle the first version of the security vault data files.
- You can enter the **--keystore-password** argument in the masked format, as shown in the following 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.

9.2.1.1.1. Migrating an Individual Security Vault to a Credential Store

Migrate individual security vaults to a credential store using the following example:

Example: Converting a single security vault to a credential store command

```
$ 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"})
```

9.2.1.1.2. Migrating 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.

Prerequisites

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**.

Procedure

1. 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"})
--------------------------------------
```

9.2.2. Migrating Security Properties to Elytron

The following examples assume that the **group.name** and **encoding.algorithm** security properties are defined as **security-properties** in the legacy **security** subsystem.

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 these 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 defines the **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 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)
```

9.3. MIGRATE AUTHENTICATION CONFIGURATION

This section provides information on migration of properties-based authentication and authorization to Elytron. In addition, it also includes information for migration of LDAP authentication, database authentication configuration, kerberos authentication, composite stores, JACC security, and security domains that use caching to Elytron.

9.3.1. Migrate PicketBox Properties-based Configuration to Elytron

Migrate PicketBox properties-based authentication to Elytron using the following example.

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>
```

9.3.1.1. Migrating Properties-based Authentication to Elytron

Follow these steps to migrate the PicketBox properties-based authentication to Elytron.

Prerequisite

The deployed web application you plan to migrate must be 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. This procedure also assumes that the security domain is defined in the legacy **security** subsystem using the following management CLI commands.

Ensure that you are starting with the PicketBox properties-based authentication configured.

Procedure

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:12.0">
  ...
  <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.

9.3.2. Migrating legacy security realm 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 in JBoss EAP 7.4 and earlier. This type of legacy security realm was typically used to secure either the management interfaces or remoting connectors.

In JBoss EAP 8.0, filesystem-realm is preferred over properties-realm.

Prerequisites

The deployed web application you plan to migrate must be 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. This procedure also assumes that the security domain is defined in the legacy **security** subsystem 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 reasons for adding the Elytron security to the application server was 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.

Procedure

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)
```

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 to secure the **http-interface** with Elytron.

```
/subsystem=elytron/http-authentication-factory=application-security-http:add(http-server-
mechanism-factory=global, security-domain=application-security, mechanism-
configurations=[{mechanism-name=BASIC}])
```

```
/core-service=management/management-interface=http-interface:write-attribute(name=http-upgrade.sasl-authentication-factory, value=application-security-sasl)
```

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 replace the names with those used in these examples when securing management interfaces.

The migration of the legacy properties-based configuration to Elytron is now complete.

9.3.3. Migrating to Filesystem-based Security Realm Using the filesystem-realm Command

Migrate the legacy properties-based security realm to a filesystem-based realm in Elytron using the **filesystem-realm** command of the **elytron.sh** tool.

A filesystem-based realm is a filesystem-based identity store used by Elytron for storing user identities. The **filesystem-realm** command converts the **properties-realm** files to **filesystem-realm**. It also generates commands for adding this realm and a security domain to the **elytron** subsystem.

Procedure

1. Migrate the properties file.
   You can migrate a single user-properties file at a time, or migrate the properties files in bulk. The following examples illustrate the procedures for both types of migration.

   - To migrate a single properties file, do this.
     The following example converts a single users-properties file with the associated roles-properties file to **filesystem-realm**. The example assumes that the legacy security domain has the following user-properties and role-properties files:

     ```
     example-users.properties
     example-roles.properties
     ```

     Example: Single user-property file migration

     ```
     $./bin/elytron-tool.sh filesystem-realm --users-file example-users.properties --roles-file example-roles.properties --output-location realms/example
     ```

     This creates filesystem-realm files and a script containing management CLI commands. The script is stored in the realms/example directory.

   - To migrate multiple properties files do.

The following example converts the users-properties files with the associated roles-properties files in bulk to **filesystem-realm**. The example assumes that the legacy security domain has the following properties files:

```
users-1.properties
users-2.properties
roles-1.properties
roles-2.properties
```

To convert users-roles files in bulk, you must create a descriptor file to use with the **filesystem-realm** command. For this example, a descriptor file **example-descriptor-file** located in the /**bin** directory is created with the following content:

Example: descriptor file

```
users-file:/full/path/to/users-1.properties
roles-file:/full/path/to/roles-1.properties
output-location:./realms/bulk-1-example
filesystem-realm-name:exampleFileSystemRealm1
security-domain-name:exampleSecurityDomain1

users-file:/full/path/to/users-2.properties
roles-file:/full/path/to/roles-2.properties
output-location:./realms/bulk-2-example
filesystem-realm-name:exampleFileSystemRealm2
security-domain-name:exampleSecurityDomain2
```

A blank line in the descriptor file is used to separate the operations for each users-properties file.

The following example converts two users-properties files with the associated roles-properties file using a descriptor file to **filesystem-realm**.

Example: Bulk migration

```
$./bin/elytron-tool.sh filesystem-realm --bulk-convert example-descriptor-file
```

This creates the **filesystem-realm** files and scripts containing the management CLI commands. The scripts are stored in directories specified in the descriptor file's **output-location** attribute.

2. Add the new security realm and the security domain to the **elytron** subsystem using the CLI commands generated by the Elytron tool.

Example: Adding the filesystem-realm

```
/subsystem=elytron/filesystem-realm=converted-properties-filesystem-
realm:add(path=/full/path/to/realms/example)

/subsystem=elytron/security-domain=converted-properties-security-domain:add(realms=
[{realm=converted-properties-filesystem-realm}],default-realm=converted-properties-
filesystem-realm,permission-mapper=default-permission-mapper)
```

9.3.4. Migrating LDAP Authentication Configuration to Elytron

Migrate legacy LDAP authentication to Elytron so that it can manage the information as identity attributes.

Prerequisites

Before you migrate legacy LDAP authentication to Elytron, you must read the content in the Migrate Properties-based Authentication and Authorization to Elytron section applies, which is also applicable here. You must focus on 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 ensure that you 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.

Procedure

1. 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.

2. 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>
```

3. 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>
```

9.3.4.1. Migrating the Legacy LDAP Authentication to Elytron

Follow these steps to migrate the previous LDAP authentication example configuration to Elytron in JBoss EAP 7.4 and earlier. This section applies to the migration of a legacy security LDAP realm as well as a PicketBox LDAP security domain.

Procedure

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.

9.3.5. Migrate Database Authentication Configuration to Elytron

Migrate JDBC datasource-based PicketBox authentication to Elytron. For instructions on defining security domains, authentication factories, and mapping them for authentication, see Migrate Properties-based Authentication and Authorization to Elytron.

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>
```

9.3.5.1. Migrating the legacy database authentication to Elytron

For JBoss EAP 7.4 and earlier releases, you must define a JDBC realm to enable JDBC datasource access by Elytron to migrate the previous database authentication example configuration to Elytron.

Procedure

1. 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.

9.3.6. 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.

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>
```

Depending on your authentication mechanisms, choose one of the following migration options:

- Migrate Kerberos HTTP Authentication
- Migrate Kerberos Remoting SASL Authentication

9.3.6.1. Migrating Kerberos HTTP Authentication

In legacy security configurations, you can define a security realm to enable SPNEGO authentication for the HTTP management interface as follows.

Prerequisite

The examples that follow assume that Kerberos is configured using the system properties.

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.

9.3.6.1.1. Migrate the Kerberos HTTP Authentication to Elytron

Secure the management interface and applications in Elytron by using a security realm and a Kerberos security factory.

Prerequisite

The examples that follow assume that Kerberos is configured using the system properties.

Procedure

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.

9.3.6.2. Migrating Kerberos Remoting SASL Authentication

Migrate Kerberos remoting SASL authentication using the following information.

Procedure

1. 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>
```

9.3.6.2.1. Migrate the Kerberos Remoting SASL Authentication to Elytron

Migrate the Kerberos remoting SASL authentication to Elytron using the following steps.

Procedure

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>
```

Verification

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.

Additional resources

These steps to define the equivalent Elytron configuration are very similar to those described in Migrate Kerberos HTTP Authentication.

9.3.7. 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 Aggregate Security Realm Configuration .

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 PicketBox Properties-based Configuration to Elytron
- Migrate LDAP Authentication Configuration to Elytron

9.3.7.1. 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>
```

For more information, see Elytron Aggregate Security Realm Configuration for how to configure an aggregate security realm.

9.3.7.2. Legacy Security Realm Composite Store Configuration

For JBoss EAP 7.4 and earlier releses, the legacy security realm configuration is configured using the following management CLI commands.

NOTE

As the legacy security commands are not applicable in JBoss EAP 8, these commands are only applicable in JBoss EAP 7.4 and earlier releases.

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.

9.3.7.3. 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.

9.3.8. Migrate security domains that use caching to Elytron

When using PicketBox, it was possible to define a security domain and enable in-memory caching for its access. This allowed you to access the identity data in memory and avoid additional direct access to the identity store. It is possible to achieve a similar configuration with Elytron. This section shows an example PicketBox configuration and equivalent security domain caching configuration when using Elytron.

9.3.8.1. PicketBox Cached Security Domain Configuration

The following commands show PicketBox security domain configuration to enable caching in JBoss EAP 7.4 and earlier.

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.

9.3.8.2. Configuring an Elytron cached security domain

Follow the steps below to create a similar configuration that caches a security domain when using Elytron.

Procedure

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

You must 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>
  ...
```

9.3.9. Migrate Jakarta authorization security to Elytron

By default, JBoss EAP 7.4 and earlier uses the legacy security subsystem to configure the Jakarta Authorization policy provider and factory. The default configuration maps to implementations from PicketBox.

The **elytron** subsystem provides a built-in policy provider based on the Java Authorization Contract for Containers (JACC) specification.

For information about how to enable and define a Java Authorization Contract for Containers policy provider in the **elytron** subsystem, see Define a Jakarta Authentication Policy Provider in the JBoss EAP 7.4 *Development Guide*.

9.4. MIGRATE APPLICATION CLIENTS

This section provides information on how to migrate client applications to Elytron.

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 example assumes 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

9.4.1. Migrating the Naming Client Using the Configuration File Approach

Migrate your naming client to Elytron using the configuration approach.

Procedure

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:1.7">
    <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");
...
```

9.4.2. Migrating 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);
```

9.4.3. Migrate a Jakarta Enterprise Beans client to Elytron

This migration example assumes that the client application is configured to invoke an Jakarta Enterprise Beans 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 Jakarta Enterprise Beans and calls one of its methods using code similar to the following example.

Example: Client code that calls a remote Jakarta Enterprise Beans

```
// 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 Jakarta Enterprise Beans 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 Jakarta Enterprise Beans client using a configuration file
- Migrate the Jakarta Enterprise Beans client programmatically

9.4.3.1. Migrate the Jakarta Enterprise Beans client using a configuration file

Follow these steps to migrate your naming client to Elytron using the configuration approach.

Procedure

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:1.7">
       <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 Jakarta Enterprise Beans 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.

9.4.3.2. Migrate the Jakarta Enterprise Beans client programmatically

Migrate the Jakarta Enterprise Beans client programmatically using the following step.

Procedure

- 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 Jakarta Enterprise Beans
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 Jakarta Enterprise Beans 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.

9.5. MIGRATE SSL CONFIGURATIONS

Migrate SSL configurations in your applications to use Elytron with the following information.

Migrate a Simple SSL Configuration to Elytron

If you secured HTTP connections to the JBoss EAP server using a security realm, migrate the SSL configuration to Elytron using the information provided in this section.

Prerequisites

Have secured HTTP connections to the JBoss EAP server using a security realm.

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>
```

Complete the following steps to achieve the same configuration using Elytron.

Procedure

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 the JBoss EAP 7.4 *How to Configure Server Security*.

9.5.1. 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

9.5.1.1. 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.

Procedure

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. Update the **https-listener** to the newly created Elytron **ssl-context**.

```
/subsystem=undertow/server=default-server/https-listener=https:write-attribute(name=ssl-
context,value=LocalhostSslContext)
```

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:14.0">
...
<https-listener name="https" socket-binding="https" ssl-context="LocalhostSslContext" enable-
http2="true"/>
...
</subsystem>
```

9.5.1.2. Security Realms and Domains

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.

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>
```

Thus, for any client certificate, a user must exist in the security realm.

9.5.1.3. 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)
```

9.5.1.4. 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>
```

9.6. LEGACY SECURITY BEHAVIOR CHANGES IN LDAP

With Red Hat JBoss Enterprise Application Platform 8.0, there are significant changes to LDAP. These changes include HTTP status change for unreachable LDAP realms, enabling LDAP security realm for role parsing from a DN, and changes in sending the JBoss EAP SSL certificate to an LDAP server.

- In JBoss EAP 8.0, which utilizes the **Elytron** subsystem, a "500 Internal Error" is returned when an LDAP realm is unreachable, indicating a HTTP status change for unreachable realms.

- In JBoss EAP 8.0, you can enable the LDAP security realm to parse roles from a DN. By loading information from LDAP as attributes associated with the identity, these attributes can subsequently be mapped to roles using the attribute-mapping element.

  Example configuration

  ```
  <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="dn" to="Roles" filter="(uniqueMember={1})" filter-base-
  dn="ou=groups,dc=group-to-principal,dc=wildfly,dc=org"/>
      </attribute-mapping>
    </identity-mapping>
  </ldap-realm>
  ```

-

- In JBoss EAP 8.0, you have the option to configure the outbound LDAP connection to utilize two-way or mutual TLS for sending the JBoss EAP SSL certificate to an LDAP server. For more information, see Enabling two-way SSL/TLS for management interfaces and applications .

APPENDIX A. REFERENCE MATERIAL

As a user of JBoss EAP, you can expect seamless compatibility and interoperability between different releases. Connecting from various clients to servers is supported, with only specific cases requiring additional considerations.

A.1. COMPATIBILITY AND INTEROPERABILITY BETWEEN RELEASES

This section describes the compatibility and interoperability of client and server enterprise beans and messaging components between the JBoss EAP 6, JBoss EAP 7, and JBoss EAP 8.0 releases.

A.1.1. Enterprise beans remoting over Internet Inter-ORB Protocol

The following configurations must run without errors:

- Connecting from a JBoss EAP 6 client to a JBoss EAP 8.0 server
- Connecting from a JBoss EAP 7 client to a JBoss EAP 8.0 server
- Connecting from a JBoss EAP 8.0 client to a JBoss EAP 7 server
- Connecting from a JBoss EAP 8.0 client to a JBoss EAP 6 server

A.1.2. Enterprise beans remoting using Java Naming and Directory Interface

The following configurations must run without errors:

- Connecting from a JBoss EAP 7 client to a JBoss EAP 8.0 server
- Connecting from a JBoss EAP 8.0 client to a JBoss EAP 7 server

JBoss EAP 6 provided support for the Enterprise Beans 3.1 specification and introduced the use of standardized global Java Naming and Directory Interface namespaces, which are still used in JBoss EAP 8.0. The changes in Java Naming and Directory Interface namespace names do not introduce incompatibilities for the following configurations:

- Connecting from a JBoss EAP 6 client to a JBoss EAP 8.0 or a JBoss EAP 7 server
- Connecting from a JBoss EAP 8.0 or JBoss EAP 7 client to a JBoss EAP 6 server

A.1.3. Enterprise beans remoting using @WebService

The following configurations must run without errors:

- Connecting from a JBoss EAP 6 client to a JBoss EAP 8.0 server
- Connecting from a JBoss EAP 7 client to a JBoss EAP 8.0 server
- Connecting from a JBoss EAP 8.0 client to a JBoss EAP 7 server
- Connecting from a JBoss EAP 8.0 client to a JBoss EAP 6 server

A.1.4. Messaging standalone client

The following configurations must run without errors:

- Connecting from a JBoss EAP 7 client to a JBoss EAP 8.0 server
- Connecting from a JBoss EAP 8.0 client to a JBoss EAP 7 server

JBoss EAP 8.0 built-in messaging is not able to connect to HornetQ 2.3.x that shipped with JBoss EAP 6 due to protocol compatibility issues. For this reason, the following configuration are not compatible:

- Connecting from a JBoss EAP 8.0 client to a JBoss EAP 6 server
- Connecting from a JBoss EAP 6 client to a JBoss EAP 8.0 server

NOTE

To make this connection possible, you must create a legacy connection factory, accessible through Java Naming and Directory Interface.

A.1.5. Messaging MDBs

The following configurations must run without errors:

- Connecting from a JBoss EAP 7 client to a JBoss EAP 8.0 server
- Connecting from a JBoss EAP 8.0 client to a JBoss EAP 7 server

JBoss EAP 8.0 built-in messaging is not able to connect to HornetQ 2.3.x that shipped with JBoss EAP 6 due to protocol compatibility issues. For this reason, the following configuration are not compatible:

- Connecting from a JBoss EAP 8.0 client to a JBoss EAP 6 server
- Connecting from a JBoss EAP 6 client to a JBoss EAP 8.0 server

NOTE

To make this connection possible, you must create a legacy connection factory, accessible through Java Naming and Directory Interface.

A.1.6. Messaging bridges

The following configurations must run without errors:

- Connecting from a JBoss EAP 6 client to a JBoss EAP 8.0 server
- Connecting from a JBoss EAP 7 client to a JBoss EAP 8.0 server
- Connecting from a JBoss EAP 8.0 client to a JBoss EAP 7 server
- Connecting from a JBoss EAP 8.0 client to a JBoss EAP 6 server