



# **JBoss Enterprise Application Platform 6.4**

## **Migrationshandbuch**

Für den Gebrauch mit der Red Hat JBoss Enterprise Application Platform 6



# JBoss Enterprise Application Platform 6.4 Migrationshandbuch

---

Für den Gebrauch mit der Red Hat JBoss Enterprise Application Platform 6

## Rechtlicher Hinweis

Copyright © 2015 Red Hat, Inc..

This document is licensed by Red Hat under the [Creative Commons Attribution-ShareAlike 3.0 Unported License](#). If you distribute this document, or a modified version of it, you must provide attribution to Red Hat, Inc. and provide a link to the original. If the document is modified, all Red Hat trademarks must be removed.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

## Zusammenfassung

Bei diesem Buch handelt es sich um ein Handbuch zur Migration früherer Versionen der Red Hat JBoss Enterprise Application Platform.

# Inhaltsverzeichnis

<b>KAPITEL 1. EINFÜHRUNG</b>	<b>5</b>
1.1. ÜBER DIE RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM 6	5
1.2. ÜBER DAS MIGRATIONSHANDBUCH	5
<b>KAPITEL 2. VORBEREITUNG DER MIGRATION</b>	<b>6</b>
2.1. VORBEREITUNG DER MIGRATION	6
2.2. WAS IST NEU UND ANDERS BEI DER JBOSS EAP 6?	6
2.3. ÜBERPRÜFEN SIE DIE LISTE VERALTETER UND NICHT UNTERSTÜTZTER FEATURES	8
<b>KAPITEL 3. MIGRIEREN SIE IHRE APPLIKATION</b>	<b>10</b>
3.1. VON DEN MEISTEN APPLIKATIONEN BENÖTIGTE ÄNDERUNGEN	10
3.1.1. Prüfung der von den meisten Applikationen benötigten Änderungen	10
3.1.2. Änderungen beim Klassenladen	10
3.1.2.1. Aktualisierung der Applikation aufgrund von Klassenladeänderungen	10
3.1.2.2. Modulabhängigkeiten verstehen	10
3.1.2.3. Aktualisierung der Applikationsabhängigkeiten aufgrund von Klassenladeänderungen	11
3.1.3. Änderungen bei Konfigurationsdateien	11
3.1.3.1. Erstellen oder Bearbeiten von Dateien, die das Klassenladen in der in der JBoss EAP 6 steuern	12
3.1.3.2. jboss-deployment-structure.xml	16
3.1.3.3. Paket-Ressourcen für das neue, modulare Klassenladesystem	16
3.1.3.4. Speicherort der ResourceBundle Properties ändern	16
3.1.3.5. Maßgeschneidertes Modul erstellen	17
3.1.4. Protokolländerungen	18
3.1.4.1. Bearbeitung von Protokollierungsabhängigkeiten	18
3.1.4.2. Aktualisierung des Applikationscodes für Protokollierungs-Frameworks von Drittanbietern	19
3.1.4.3. Bearbeitung des Codes zur Verwendung des neuen JBoss Logging Framework	21
3.1.5. Änderungen beim Packen der Applikation	22
3.1.5.1. Modifizierung des Packagings von EARs und WARs	22
3.1.6. Konfigurationsänderungen bei Datenquellen und Ressourcenadaptern	22
3.1.6.1. Aktualisierung der Applikation aufgrund von Konfigurationsänderungen	22
3.1.6.2. Aktualisierung der Datenquellen-Konfiguration	23
3.1.6.3. Installation und Konfiguration des JDBC-Treibers	24
3.1.6.4. Konfiguration der Datenquelle für Hibernate oder JPA	29
3.1.6.5. Aktualisierung der Ressourcen-Adapter-Konfiguration	29
3.1.7. Sicherheitsänderungen	30
3.1.7.1. Konfiguration der Sicherheitsänderungen der Applikation	30
3.1.7.2. Aktualisieren von Applikationen, die PicketLink STS und Web Services benutzen	31
3.1.8. JNDI-Änderungen	32
3.1.8.1. Aktualisierung der JNDI Namespace-Namen der Applikation	32
3.1.8.2. Portierbare EJB JNDI-Namen	33
3.1.8.3. Übersicht über die JNDI-Namespaces Regeln	34
3.1.8.4. Bearbeitung der Applikation zur Befolgung der neuen JNDI-Namespaces Regeln	34
3.1.8.5. Beispiele von JNDI-Namespaces in früheren Releases und wie diese in der JBoss EAP 6 spezifiziert sind	35
3.1.9. HTTP/HTTPS/AJP Konnektorattribute zuordnen	36
3.1.9.1. Zuordnen der HTTP/HTTPS/AJP Connector Attribute	36
3.2. VON IHRER APPLIKATIONSARCHITEKTUR UND -KOMPONENTEN ABHÄNGIGE ÄNDERUNGEN	41
3.2.1. Übersicht der von Ihrer Applikationsarchitektur und -komponenten abhängigen Änderungen	41
3.2.2. Hibernate- und JPA-Änderungen	42
3.2.2.1. Aktualisierung von Applikationen, die Hibernate und/oder JPA verwenden	42
3.2.2.2. Konfiguration der Änderungen bei Applikationen, die Hibernate und JPA verwenden	42
3.2.2.3. Properties der Persistenzeinheit	44

3.2.2.4. Aktualisieren Sie Ihre Hibernate 3 Applikation, damit sie Hibernate 4 benutzt	46
3.2.2.5. Beibehaltung des bestehenden Verhaltens des selbstgenerierten Wertes der Hibernate-Identität	46
3.2.2.6. Migration Ihrer Hibernate 3.3.x Applikation zu Hibernate 4.x	47
3.2.2.7. Migration Ihrer Hibernate 3.5.x Applikation zu Hibernate 4.x	48
3.2.2.8. Bearbeiten Sie Persistenz-Properties für migrierte Seam- und Hibernate-Applikationen, die in einer geclusterten Umgebung laufen	49
3.2.2.9. Aktualisieren Sie Ihre Applikation, damit sie mit der JPA 2.0 Spezifikation konform ist	49
3.2.2.10. Ersetzen des Caches der zweiten Ebene von JPA/Hibernate durch Infinispan	50
3.2.2.11. Hibernate Cache Properties	51
3.2.2.12. Migration zu Hibernate Validator 4	52
3.2.3. JSF-Änderungen	54
3.2.3.1. Applikationen die Verwendung älterer Versionen von JSF ermöglichen	54
3.2.4. Web-Dienste Änderungen	54
3.2.4.1. Web-Dienste Änderungen	54
3.2.5. JAX-RS- und RESTEasy-Änderungen	57
3.2.5.1. Konfiguration der JAX-RS- und RESTEasy-Änderungen	57
3.2.6. Änderungen des LDAP-Sicherheitsbereichs	59
3.2.6.1. Konfiguration der Änderungen des LDAP-Sicherheitsbereichs	59
3.2.7. HornetQ-Änderungen	60
3.2.7.1. Über HornetQ und NFS	60
3.2.7.2. Konfiguration einer JMS-Bridge zur Migration bestehender JMS-Nachrichten zur JBoss EAP 6	60
3.2.7.3. JMS Bridge erstellen	61
3.2.7.4. Migration Ihrer Applikation zur Verwendung von HornetQ als dem JMS-Provider	66
3.2.7.5. Konfigurieren Nachrichten-Vermittlung für HornetQ	67
3.2.8. Clustering-Änderungen	67
3.2.8.1. Durchführung von Änderungen an Ihrer Applikation für Clustering	67
3.2.8.2. Implementierung eines HA-Singleton	72
3.2.9. Änderungen beim Deployment im Dienst-Stil	78
3.2.9.1. Aktualisierung von Applikationen, die Deployments im Dienst-Stil verwenden	78
3.2.10. Änderungen bei Remote-Aufrufen	78
3.2.10.1. Migration von JBoss EAP 5 deployten Applikationen, die Remote-Aufrufe zur JBoss EAP 6 tätigen	78
3.2.10.2. Remote-Aufruf eines Session-Beans mittels JNDI	80
3.2.10.3. EJB JNDI Namensgebungsreferenz	83
3.2.11. EJB 2.x Änderungen	84
3.2.11.1. Aktualisierung von Applikationen, die EJB 2.x verwenden	84
3.2.12. JBoss AOP Änderungen	91
3.2.12.1. Aktualisierung von Applikationen, die JBoss AOP verwenden	91
3.2.13. Migration von Seam 2.2 Applikationen	91
3.2.13.1. Migration der Seam 2.2 Archive zur JBoss EAP 6	91
3.2.13.2. Seam 2.2 Archiv-Migrationsprobleme	95
3.2.14. Migration von Spring Applikationen	98
3.2.14.1. Migration von Spring Applikationen	98
3.2.15. Andere Änderungen, die Ihre Migration beeinflussen können	98
3.2.15.1. Machen Sie sich mit anderen Änderungen vertraut, die Einfluss auf Ihre Migration haben können	98
3.2.15.2. Änderung des Maven Plug-in Namens	98
3.2.15.3. Bearbeitung von Client-Applikationen	98

## **KAPITEL 4. TOOLS UND TIPPS ..... 100**

4.1. RESSOURCEN, DIE BEI DER MIGRATION HELFEN SOLLEN	100
4.1.1. Ressourcen, die bei Ihrer Migration helfen sollen	100
4.1.2. Machen Sie sich mit den Tools vertraut, die Ihnen bei der Migration helfen können	100

4.1.3. Verwendung von Tattletale zum Auffinden von Applikationsabhängigkeiten	101
4.1.4. Download und Installation von Tattletale	101
4.1.5. Erstellen und Prüfen des Tattletale Berichts	101
4.1.6. Verwendung des IronJacamar Migrationstools zur Migration der Datenquellen- und Ressourcenadapterkonfigurationen	102
4.1.7. Download und Installation des IronJacamar Migrationstools	102
4.1.8. Verwenden Sie das IronJacamar Migrationstool für die Konvertierung einer Datenquellen-Konfigurationsdatei	103
4.1.9. Verwenden Sie das IronJacamar Migrationstool für die Konvertierung einer Ressourcenadapter-Konfigurationsdatei	105
4.2. FEHLERBEHEBUNG BEI DER MIGRATION	110
4.2.1. Fehler- und Problembehebung bei der Migration	110
4.2.2. Fehlerbehebung und Auflösung von ClassNotFoundExceptions und NoClassDefFoundErrors	110
4.2.3. Auffinden der JBoss Modulabhängigkeit	110
4.2.4. Finden Sie das JAR in der vorherigen Installation	111
4.2.5. Fehler- und Problembehebung von ClassCastExceptions	112
4.2.6. Fehlerbehebung und Auflösung von DuplicateServiceExceptions	113
4.2.7. Fehlerbehebung und Auflösung von JBoss Seam Debug-Seitenfehlern	113
4.3. ÜBERSICHT DER MIGRATION DER BEISPIELAPPLIKATIONEN	115
4.3.1. Übersicht der Migration der Beispielapplikationen	116
4.3.2. Migration des Seam 2.2 JPA Beispiels zur JBoss EAP 6	116
4.3.3. Migration des Seam 2.2 Buchungsbeispiels zur JBoss EAP 6	117
4.3.4. Migration des Seam 2.2 Buchungsarchivs zur JBoss EAP 6: Schritt-für-Schritt Anleitung	121
4.3.5. Erstellen und Deployment der JBoss EAP 5.X Version der Seam 2.2 Booking Applikation	122
4.3.6. Fehlerbehebung und Auflösung von Seam 2.2 Booking Archive Deployment Fehlern und Ausnahmen	123
4.3.7. Fehlerbehebung und Auflösung von Seam 2.2 Booking Archive Runtime Fehlern und Ausnahmen	131
4.3.8. Übersicht über eine Zusammenfassung von Änderungen bei der Migration der Seam 2.2 Booking Application	135
<b>ANHANG A. VERSIONSGESCHICHTE</b>	<b>138</b>





# KAPITEL 1. EINFÜHRUNG

## 1.1. ÜBER DIE RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM 6

Die Red Hat JBoss Enterprise Application Platform 6 (JBoss EAP 6) ist eine Middleware-Plattform, die auf offenen Standards basiert und mit der Java Enterprise Edition 6 Spezifikation konform ist. Sie integriert den JBoss Application Server 7 mit High Availability Clustering, Messaging, verteiltem Caching und anderen Technologien.

Die JBoss EAP 6 enthält eine neue, modulare Struktur, die es ermöglicht, Dienste erst bei Bedarf zu aktivieren und so den Systemstart zu beschleunigen.

Die Management-Konsole und das Management Command Line Interface machen das Bearbeiten von XML-Konfigurationsdateien überflüssig und ermöglichen die Verwendung von Skripten und die Automatisierung von Aufgaben.

Des Weiteren beinhaltet die JBoss EAP 6 APIs und Development Frameworks zur schnellen Entwicklung sicherer und skalierbarer Java EE Anwendungen.

[Fehler melden](#)

## 1.2. ÜBER DAS MIGRATIONSHANDBUCH

Bei der JBoss EAP 6 handelt es sich um eine schnelle, leichtgewichtige und leistungsfähige Implementierung der Java Enterprise Edition 6 Spezifikation. Die Architektur ist auf dem Modular Service Container (Modulardienst-Container) erbaut und aktiviert Dienste auf Abruf, wenn Ihre Applikation diese benötigt. Aufgrund dieser neuen Architektur ist es möglich, dass Applikationen, die auf der JBoss EAP 5 laufen bearbeitet werden müssen, um auf der JBoss EAP 6 zu laufen.

Dieses Handbuch dokumentiert die Änderungen, die notwendig sind, damit JBoss EAP 5.1 Applikationen erfolgreich auf der EAP 6 laufen und deployt werden können. Es liefert Informationen zur Lösung von Problemen beim Deployment und zur Runtime sowie dazu, wie sich Änderungen im Anwendungsverhalten vermeiden lassen. Dies ist der erste Schritt in Richtung einer neuen Plattform. Wurde die Applikation erfolgreich deployt und läuft diese, so können Sie mit der Planung eines Upgrades einzelner Komponenten beginnen, um die neuen Funktionen und Features der JBoss EAP 6 zu nutzen.

[Fehler melden](#)

## KAPITEL 2. VORBEREITUNG DER MIGRATION

### 2.1. VORBEREITUNG DER MIGRATION

Da der Applikationsserver anders als in früheren Versionen strukturiert ist, sollten Sie sich ausreichend informieren und gründlich planen, ehe Sie mit der Migration Ihrer Applikation beginnen.

#### 1. Was ist neu und anders bei der JBoss EAP 6?

Eine Reihe von Dingen hat sich in dieser Release geändert und kann Auswirkungen auf das Deployment von Applikationen der JBoss EAP 5 haben. Dies beinhaltet Änderungen am Dateiverzeichnis, Skripten, der Deployment-Konfiguration, dem Klassenladen und JNDI-Lookups. Weitere Informationen finden Sie unter [Abschnitt 2.2, »Was ist neu und anders bei der JBoss EAP 6?«](#).

#### 2. Sehen Sie sich die "Get Started"-Dokumentation noch einmal an

Sehen Sie sich das Kapitel mit dem Titel *Get Started Developing Applications* im *Development Guide* für die JBoss EAP 6 unter

[https://access.redhat.com/site/documentation/JBoss\\_Enterprise\\_Application\\_Platform/](https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/) an. Es enthält wichtige Informationen zu dem Folgenden:

- Java EE 6
- Das neue, modulare Klassenladesystem
- Änderungen an der Dateistruktur
- Download und Installation der JBoss EAP 6
- Download und Installation des JBoss Developer Studio
- Konfiguration von Maven für Ihre Entwicklungsumgebung
- Download und Betrieb des mit dem Produkt gelieferten Quickstart-Beispiels.

#### 3. Verwendung der JBoss EAP 6 Abhängigkeiten in Ihrem Maven Projekt

Sehen Sie sich das Kapitel *Maven Guide* im *Development Guide* für JBoss EAP 6 unter [https://access.redhat.com/site/documentation/JBoss\\_Enterprise\\_Application\\_Platform/](https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/) an. Der *Manage Project Dependencies* Abschnitt enthält wichtige Informationen darüber, wie Ihr Projekt konfiguriert wird um JBoss EAP Bill of Material (BOM) Artefakte zu benutzen.

#### 4. Analyse und Verständnis Ihrer Applikation

Jede Applikation ist einzigartig und Sie sollten die Komponenten und die Architektur der bestehenden Applikation verstehen, ehe Sie versuchen, die Migration durchzuführen.



#### WICHTIG

Ehe Sie Änderungen an Ihrer Applikation vornehmen, sollten Sie auf jeden Fall eine Backup-Kopie erstellen.

[Fehler melden](#)

### 2.2. WAS IST NEU UND ANDERS BEI DER JBOSS EAP 6?

#### Einführung

Nachfolgend sehen Sie eine Liste von Unterschieden zwischen der JBoss EAP 6 und der früheren Release.

### Modulbasiertes Klassenladen

In der JBoss EAP 5 war die Klassenladearchitektur hierarchisch. In der JBoss EAP 6 basiert das Laden von Klassen auf JBoss Modulen. Dies bietet echte Applikationsisolation, verbirgt Serverimplementierungsklassen und lädt nur diejenigen Klassen, die Ihre Applikation benötigt. Für eine bessere Performance erfolgt das Klassenladen nebenläufig. Für die JBoss EAP 5 geschriebene Applikationen müssen bearbeitet werden, um Modulabhängigkeiten festzulegen und Archive müssen - in manchen Fällen - neu gepackt werden. Weitere Informationen finden Sie unter *Klassenladen und Moduledes Entwicklungshandbuchs* für die JBoss EAP 6 unter [https://access.redhat.com/site/documentation/JBoss\\_Enterprise\\_Application\\_Platform/](https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/).

### Domain-Management

Bei der JBoss EAP 6 kann der Server als ein Standalone Server oder in einer Managed Domain ausgeführt werden. In einer Managed Domain können Sie ganze Gruppen von Servern gleichzeitig konfigurieren, so dass Konfigurationen über Ihr gesamtes Netzwerk an Servern hinweg synchronisiert bleiben. Während dies keine Auswirkungen auf für frühere Releases erstellte Applikationen haben sollte, kann es das Management von Deployments auf mehreren Servern vereinfachen. Weitere Informationen finden Sie in *About Managed Domains* im *Administration and Configuration Guide* für die JBoss EAP 6 unter [https://access.redhat.com/site/documentation/JBoss\\_Enterprise\\_Application\\_Platform/](https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/).

### Deployment-Konfiguration

#### Standalone Server und Managed Domains

Die JBoss EAP 5 verwendete eine profilbasierte Deployment-Konfiguration. Diese Profile befanden sich im **EAP\_HOME/server/-**Verzeichnis. Applikationen enthielten oft mehrere Konfigurationsdateien für Sicherheit, Datenbank, Ressourcenadapter, und andere Konfigurationen. Bei der JBoss EAP 6 erfolgt die Deployment-Konfiguration mittels einer Datei. Diese wird zur Konfiguration aller für das Deployment verwendeten Dienste und Subsysteme benutzt. Ein Standalone Server wird mittels der **EAP\_HOME/standalone/configuration/standalone.xml**-Datei konfiguriert. In einer Managed Domain laufende Server werden mittels der **EAP\_HOME/domain/configuration/domain.xml**-Datei konfiguriert. Die auf mehrere Konfigurationsdateien der JBoss EAP 5 verteilten Informationen müssen in eine einzelne neue Konfigurationsdatei migriert werden.

#### Reihenfolge der Deployments

Die JBoss Enterprise Application Platform 6 verwendet schnelle, nebenläufige Initialisierung für das Deployment, was zu einer besseren Performance und Effizienz führt. In den meisten Fällen kann der Applikationsserver Abhängigkeiten vorab automatisch bestimmen und dann die effizienteste Deployment-Strategie wählen. JBoss Enterprise Application Platform 5 Applikationen hingegen, die aus mehreren Modulen bestehen, die als EARs deployt werden und JNDI-Lookups statt CDI-Einspeisung oder resource-ref Einträgen verwenden, können Konfigurationsänderungen benötigen.

### Verzeichnisstruktur und Skripte

Wie bereits erwähnt verwendet die JBoss EAP 6 keine profilbasierte Deployment-Konfiguration mehr, weshalb auch kein **EAP\_HOME/server/-**Verzeichnis benötigt wird. Konfigurationsdateien für Standalone Server befinden sich jetzt im **EAP\_HOME/standalone/configuration/-**Verzeichnis,

und Deployments befinden sich im **`EAP_HOME/standalone/deployments/`**-Verzeichnis. Die Konfigurationsdateien von in einer Managed Domain laufenden Servern finden Sie im **`EAP_HOME/domain/configuration/`**-Verzeichnis.

Bei der JBoss EAP 5 wurden das Linux-Skript **`EAP_HOME/bin/run.sh`** oder das Windows-Skript **`EAP_HOME/bin/run.bat`** zum Starten des Servers verwendet. Bei der JBoss EAP 6 ist das Server-Startskript abhängig davon, wie Sie Ihren Server betreiben. Das Linux-Skript **`EAP_HOME/bin/standalone.sh`** oder das Windows-Skript **`EAP_HOME/bin/standalone.bat`** werden für den Start eines Standalone Servers verwendet. Das Linux-Skript **`EAP_HOME/bin/domain.sh`** oder das Windows-Skript **`EAP_HOME/bin/domain.bat`** werden beim Start einer Managed Domain eingesetzt.

## JNDI-Lookups

Die JBoss EAP 6 verwendet jetzt standardisierte portierbare JNDI-Namespace. Für die JBoss EAP 5 geschriebene Applikationen, die JNDI-Lookups verwenden, müssen geändert werden, damit sie der neuen, standardisierten JNDI-Namespace Konvention folgen. Weitere Informationen zur Syntax der JNDI-Namensgebung finden Sie unter [Abschnitt 3.1.8.2, »Portierbare EJB JNDI-Namen«](#).

Weitere Informationen finden Sie unter *New and Changed Features in JBoss EAP 6* im Kapitel des *Entwicklungshandbuchs* für die JBoss EAP 6 unter [https://access.redhat.com/site/documentation/JBoss\\_Enterprise\\_Application\\_Platform/](https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/).

[Fehler melden](#)

## 2.3. ÜBERPRÜFEN SIE DIE LISTE VERALTETER UND NICHT UNTERSTÜTZTER FEATURES

Bevor Sie Ihre Applikation migrieren, sollten Sie sich bewusst sein, dass einige Features, die in früheren Versionen der JBoss EAP verfügbar waren, vielleicht veraltet sind oder nicht mehr unterstützt werden. Eine umfassende Liste finden Sie im *Unsupported Features* Abschnitt der *Release Notes* für JBoss EAP 6 im Kundenportal unter

[https://access.redhat.com/site/documentation/JBoss\\_Enterprise\\_Application\\_Platform/](https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/).

Es folgt eine kurze Zusammenfassung einiger nicht unterstützter Features.

### Server Startup -b Befehlszeilenargument

In früheren Versionen verwendete JBoss EAP automatisch die vom **`-b`** Startup Parameter definierten Adressen, ungeachtet der IP-Adresse. In JBoss EAP 6 sucht die Server **`<inet-address>`** Konfiguration nach einem Netzwerk Interface mit passender IP-Adresse. Während das für **`127.0.0.1`** funktioniert, funktioniert es für **`127.*.*.*`** IP-Adressen nicht mehr. Wenn Sie den JBoss EAP 6 Server mit dem **`-b`** Befehlszeilenargument zur Bindung an **`127.*.*.*`** IP-Adressen starten, müssen Sie nun zuerst das Interface von **`<inet-address>`** zu **`<loopback-address>`** in der Server-Konfigurationsdatei ändern.

Weitere Information darüber, wie man den Server unter Verwendung des Management CLI konfiguriert, finden Sie im Abschnitt *Management CLI Operations* im *Administration and Configuration Guide* für JBoss Enterprise Application Platform im Kundenportal unter [https://access.redhat.com/site/documentation/JBoss\\_Enterprise\\_Application\\_Platform/](https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/).

### EJB Abhängigkeiten

In früheren Versionen der JBoss EAP konnten EJB Abhängigkeiten von einem Dienst oder Diensten, einschließlich anderer EJBs, unter Verwendung eines **<depends>** Tags im **jboss.xml** Deployment Deskriptor angegeben werden. Zum Beispiel:

```
<depends>jboss.j2ee:jndiName=com/myorg/app/Foo,service=EJB</depends>
<depends>jboss.mq.destination:service=Queue,name=queue/HelloworldQueue</depends>
```

Bei der JBoss EAP 6 müssen Sie die **@EJB** Annotation benutzen um EJB Referenzen einzuspeisen und die **@Resource** Annotation benutzen um auf Datenquellen oder andere Ressourcen zuzugreifen. Zum Beispiel:

```
@EJB(lookup="java:global/MyApp/FooImpl!com.myorg.app.Foo")
@Resource(mappedName = "java:/queue/HelloworldQueue")
```

JNDI Lookups wurden ebenfalls geändert. Im *JNDI Changes* Abschnitt dieses Handbuchs finden Sie weitere Details.

Weitere Information über EJB Referenzen finden Sie im *EJB Reference Resolution* Abschnitt des *Development Guide* for JBoss Enterprise Application Platform im Kundenportal unter [https://access.redhat.com/site/documentation/JBoss\\_Enterprise\\_Application\\_Platform/](https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/).

## HTTPInvoker

In früheren Versionen der JBoss EAP war es möglich den HTTPInvoker zur Konfiguration von EJB, JNDI oder JMS zu benutzen um das HTTP Protokoll zu verwenden. Dies ist in JBoss EAP 6 nicht mehr möglich.

## HA Singleton Deployments und BarrierController Service

Der HA SingletonService garantiert, dass es nur eine Instanz eines im Cluster laufenden Dienstes gibt.

JBoss EAP 5 bietet Support für mehrere Strategien für HA Singleton Deployments, einschließlich des HASingletonDeployer Services, HASingletonController benutzende POJO Deployments und BarrierController Service benutzende HASingleton Deployments. All diese Strategien sind auf HAPartition angewiesen um Meldungen auszugeben, wenn verschiedene Knoten im Cluster gestartet und angehalten werden, und sie sind nicht mehr verfügbar.

Bei der JBoss EAP 6 wurden HA Singleton Deployments vollständig verändert. Der Singleton Deployer arbeitet jetzt nur auf Modular Service Container (MSC) Diensten. Wenn man einen SingletonService benutzt, ist der Zieldienst an jedem Knoten im Cluster installiert, wird aber zu jeder Zeit nur an einem Knoten gestartet. Dieser Ansatz vereinfacht die Deployment Anforderungen und minimiert die erforderliche Zeit um den Singleton Master Service zwischen Knoten zu verschieben. Es ist jedoch erforderlich, dass Sie benutzerdefinierte Codes schreiben um die gleiche Funktionalität zu erzielen. Ein Beispiel für ein HA Singleton Deployment ist in den JBoss EAP Quickstart Beispielapplikationen enthalten, die mit dem Produkt geliefert werden. Weitere Informationen zu HA Singletons finden Sie unter [Abschnitt 3.2.8.2, »Implementierung eines HA-Singleton«](#).

## Fehler melden

## KAPITEL 3. MIGRIEREN SIE IHRE APPLIKATION

### 3.1. VON DEN MEISTEN APPLIKATIONEN BENÖTIGTE ÄNDERUNGEN

#### 3.1.1. Prüfung der von den meisten Applikationen benötigten Änderungen

Klassenlade- und Konfigurationsänderungen in der JBoss EAP 6 haben Einfluss auf fast jede Applikation. Die JBoss EAP 6 verwendet außerdem standardmäßig portierbare JNDI-Namensgebungssyntax. Diese Änderungen betreffen fast alle Applikationen, so dass wir empfehlen, dass Sie die nachfolgenden Informationen lesen, ehe Sie mit der Migration Ihrer Applikation beginnen.

1. [Abschnitt 3.1.2.1, »Aktualisierung der Applikation aufgrund von Klassenladeänderungen«](#)
2. [Abschnitt 3.1.6.1, »Aktualisierung der Applikation aufgrund von Konfigurationsänderungen«](#)
3. [Abschnitt 3.1.8.1, »Aktualisierung der JNDI Namespace-Namen der Applikation«](#)

[Fehler melden](#)

#### 3.1.2. Änderungen beim Klassenladen

##### 3.1.2.1. Aktualisierung der Applikation aufgrund von Klassenladeänderungen

Beim modularen Laden von Klassen handelt es sich um eine maßgebliche Änderung in der JBoss EAP 6, die auf fast alle Applikationen Auswirkungen hat. Sehen Sie sich zunächst die folgenden Informationen an, ehe Sie Ihre Applikation migrieren.

1. Sehen Sie sich zuerst an, wie Ihr Applikation gepackt ist und welche Abhängigkeiten Sie besitzt. Weitere Informationen finden Sie hier [Abschnitt 3.1.2.3, »Aktualisierung der Applikationsabhängigkeiten aufgrund von Klassenladeänderungen«](#)
2. Falls Ihre Applikation ein Protokoll führt, so müssen Sie die korrekten Modulabhängigkeiten festlegen. Unter [Abschnitt 3.1.4.1, »Bearbeitung von Protokollierungsabhängigkeiten«](#) erfahren Sie mehr dazu.
3. Aufgrund der Änderungen beim modularen Klassenladen kann es sein, dass Sie die Packstruktur Ihres EAR oder WAR ändern müssen. Weitere Informationen finden Sie unter [Abschnitt 3.1.5.1, »Modifizierung des Packagings von EARs und WARs«](#).

[Fehler melden](#)

##### 3.1.2.2. Modulabhängigkeiten verstehen

###### Zusammenfassung

Ein Modul kann nur auf seine eigenen Klassen zugreifen sowie auf die Klassen eines Moduls von dem es eine implizite oder explizite Abhängigkeit besitzt.

###### Prozedur 3.1. Modulabhängigkeiten verstehen

###### 1. Implizite Modulabhängigkeiten verstehen

Die Deployer innerhalb des Servers fügen implizit automatisch einige häufig verwendete Modulabhängigkeiten hinzu, wie etwa `javax.api` und `sun.jdk`. Dies macht die Klassen für das Deployment zur Runtime sichtbar, so dass der Entwickler die Abhängigkeiten nicht explizit

hinzufügen muss. Informationen dazu, wie und wann diese impliziten Abhängigkeiten hinzugefügt werden, finden Sie unter *Implicit Module Dependencies* im Kapitel *Class Loading and Modules* des *Development Guide* für die JBoss EAP 6 unter [https://access.redhat.com/site/documentation/JBoss\\_Enterprise\\_Application\\_Platform/](https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/).

## 2. Explizite Modulabhängigkeiten verstehen

Für andere Klassen müssen Module explizit festgelegt werden, da die fehlenden Abhängigkeiten ansonsten zu Deployment- oder Runtime-Fehlern führen. Falls eine Abhängigkeit fehlt, so sehen Sie **ClassNotFoundExceptions** oder **NoClassDefFoundErrors**-Traces im Serverprotokoll. Lädt mehr als ein Modul dasselbe JAR oder ein Modul lädt eine Klasse, die eine von einem anderen Modul geladene Klasse erweitert, so sehen Sie **ClassCastExceptions**-Traces im Serverprotokoll. Um Abhängigkeiten explizit festzulegen, bearbeiten Sie **MANIFEST.MF** oder erstellen Sie eine JBoss-spezifische Deployment-Deskriptor-Datei **jboss-deployment-structure.xml**. Weitere Informationen zu Modulabhängigkeiten finden Sie unter *Overview of Class Loading and Modules* im Kapitel *Class Loading and Modules* des *Development Guide* für die JBoss EAP 6 unter [https://access.redhat.com/site/documentation/JBoss\\_Enterprise\\_Application\\_Platform/](https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/).

[Fehler melden](#)

### 3.1.2.3. Aktualisierung der Applikationsabhängigkeiten aufgrund von Klassenladeänderungen

#### Zusammenfassung

Das Laden von Klassen unterscheidet sich bei der JBoss EAP 6 maßgeblich von früheren Versionen der JBoss EAP. Das Klassenladen basiert jetzt auf dem JBoss Modulprojekt. Statt eines einzelnen, hierarchischen Klassenladers, der alle JARs in einen flachen Klassenpfad lädt, wird jede Bibliothek zu einem Modul, das sich nur mit dem Modul verbindet, von dem es abhängt. Deployments in der JBoss EAP 6 sind ebenfalls Module und haben keinen Zugriff auf in JARs im Applikationsserver definierten Klassen, außer es wurde eine explizite Abhängigkeit von diesen Klassen definiert. Einige vom Applikationsserver definierte Modulabhängigkeiten werden automatisch für Sie eingestellt. Wenn Sie etwa eine Java EE Applikation deployen, so wird Ihrem Modul automatisch eine Abhängigkeit vom Java EE API hinzugefügt. Eine vollständige Liste automatisch hinzugefügter Abhängigkeiten finden Sie unter *Implicit Module Dependencies* im Kapitel *Class Loading and Modules* des *Development Guide* für die JBoss EAP 6 unter [https://access.redhat.com/site/documentation/JBoss\\_Enterprise\\_Application\\_Platform/](https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/).

#### Aufgaben

Wenn Sie Ihre Applikation zur JBoss EAP 6 migrieren, so kann es sein, dass Sie aufgrund der modularen Klassenladeänderungen eine der folgenden Aufgaben ausführen müssen:

- [Abschnitt 3.1.2.2, »Modulabhängigkeiten verstehen«](#)
- [Abschnitt 4.1.3, »Verwendung von Tattletale zum Auffinden von Applikationsabhängigkeiten«](#)
- [Abschnitt 3.1.3.1, »Erstellen oder Bearbeiten von Dateien, die das Klassenladen in der in der JBoss EAP 6 steuern«](#)
- [Abschnitt 3.1.3.3, »Paket-Ressourcen für das neue, modulare Klassenladesystem«](#)

[Fehler melden](#)

### 3.1.3. Änderungen bei Konfigurationsdateien



### 3.1.3.1. Erstellen oder Bearbeiten von Dateien, die das Klassenladen in der in der JBoss EAP 6 steuern

#### Zusammenfassung

Aufgrund der Änderungen in der JBoss EAP 6 zur Verwendung modularen Klassenladens, müssen Sie möglicherweise eine oder mehrere Dateien bearbeiten oder erstellen, um Abhängigkeiten hinzuzufügen oder das Laden automatischer Abhängigkeiten zu verhindern. Weitere Informationen zum Laden von Klassen oder Klassenladepräzedenz finden Sie im Kapitel *Class Loading and Modules* des *Development Guide* für die JBoss Enterprise Application Platform 6 unter [https://access.redhat.com/site/documentation/JBoss\\_Enterprise\\_Application\\_Platform/](https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/).

Die folgenden Dateien werden zur Steuerung des Ladens von Klassen in der JBoss EAP 6 verwendet.

#### **jboss-web.xml**

Falls Sie ein **<class-loading>**-Element in der **jboss-web.xml**-Datei definiert haben, so müssen Sie dieses entfernen. Das Verhalten, das dies in der JBoss EAP 5 hervorrief, ist jetzt das standardmäßige Klassenladeverhalten in der JBoss EAP 6, so dass dies nicht länger notwendig ist. Falls Sie dieses Element nicht entfernen, so erscheint im Serverprotokoll ein `ParseError` und eine `XMLStreamException`.

Dies ist ein Beispiel eines herauskommentierten **<class-loading>**-Elements in der **jboss-web.xml**-Datei.

```
<!DOCTYPE jboss-web PUBLIC
    "-//JBoss//DTD Web Application 4.2//EN"
    "http://www.jboss.org/j2ee/dtd/jboss-web_4_2.dtd">
<jboss-web>
<!--
    <class-loading java2ClassLoadingCompliance="false">
        <loader-repository>
            seam.jboss.org:loader=MyApplication
        </loader-repository>
    </class-loading>
-->
</jboss-web>
```

#### **MANIFEST.MF**

##### **Manuell bearbeitet**

Je nachdem, welche Komponenten oder Module Ihre Applikation verwendet, werden Sie dieser Datei eine oder mehrere Abhängigkeiten hinzufügen müssen. Sie können diese entweder als **Dependencies**- oder **Class-Path**-Einträge hinzufügen.

Nachfolgend sehen Sie ein von einem Entwickler bearbeitetes **MANIFEST.MF**:

```
Manifest-Version: 1.0
Dependencies: org.jboss.logmanager
Class-Path: OrderManagerEJB.jar
```



Falls Sie diese Datei bearbeiten, vergessen Sie bitte nicht, ein "Newline"-Zeichen am Ende der Datei hinzuzufügen.

### Mittels Maven generiert

Falls Sie Maven verwenden, so müssen Sie Ihre **pom.xml**-Datei bearbeiten, um die Abhängigkeiten für die **MANIFEST.MF**-Datei zu generieren. Falls Ihre Applikation EJB 3.0 verwendet, so haben Sie vielleicht einen Abschnitt in der **pom.xml**-Datei, der wie folgt aussieht:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-ejb-plugin</artifactId>
  <configuration>
    <ejbVersion>3.0</ejbVersion>
  </configuration>
</plugin>
```

Falls der EJB 3.0 Code **org.apache.commons.log** verwendet, so benötigen Sie diese Abhängigkeit in der **MANIFEST.MF**-Datei. Um diese Abhängigkeit zu generieren, fügen Sie das **<plugin>**-Element wie folgt der **pom.xml**-Datei hinzu:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-ejb-plugin</artifactId>
  <configuration>
    <ejbVersion>3.0</ejbVersion>
    <archive>
      <manifestFile>src/main/resources/META-
INF/MANIFEST.MF</manifestFile>
    </archive>
  </configuration>
</plugin>
```

Im Beispiel oben muss die **src/main/resources/META-INF/MANIFEST.MF**-Datei nur den folgenden Abhängigkeitseintrag enthalten:

```
Dependencies: org.apache.commons.logging
```

Maven generiert die vollständige **MANIFEST.MF**-Datei:

```
Manifest-Version: 1.0
Dependencies: org.apache.commons.logging
```

### jboss-deployment-structure.xml

Bei dieser Datei handelt es sich um einen für JBoss spezifischen Deployment-Deskriptor, der zur Steuerung des Klassenladens auf feinkörnige Weise verwendet werden kann. Wie auch **MANIFEST.MF** kann diese Datei für die Hinzufügung von Abhängigkeiten verwendet werden. Sie kann auch die Hinzufügung automatischer Abhängigkeiten verhindern, zusätzliche Module definieren, das isolierte Klassenladeverhalten eines EAR-Deployments ändern und einem Modul zusätzliche Ressourcen-roots hinzufügen.

Nachfolgend sehen Sie das Beispiel einer **jboss-deployment-structure.xml**-Datei, die eine Abhängigkeit für das JSF 1.2 Modul hinzufügt und das automatische Laden für das JSF 2.0 verhindert.

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.0">
  <deployment>
    <dependencies>
      <module name="javax.faces.api" slot="1.2" export="true"/>
      <module name="com.sun.jsf-impl" slot="1.2" export="true"/>
    </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
      <module name="javax.faces.api" slot="main"/>
      <module name="com.sun.jsf-impl" slot="main"/>
    </exclusions>
    <dependencies>
      <module name="javax.faces.api" slot="1.2"/>
      <module name="com.sun.jsf-impl" slot="1.2"/>
    </dependencies>
  </sub-deployment>
</jboss-deployment-structure>
```

Weitere Informationen zu dieser Datei finden Sie unter: [Abschnitt 3.1.3.2, »jboss-deployment-structure.xml«](#).

## application.xml

In früheren Versionen der JBoss EAP wurde die Reihenfolge der Deployments innerhalb eines EAR mittels der **jboss-app.xml**-Datei gesteuert. Dies ist nicht mehr der Fall. Die Java EE6 Spec liefert das **<initialize-in-order>**-Element in der **application.xml**, mit der die Reihenfolge des Deployments der Java EE Module innerhalb eines EAR gesteuert wird.

In den meisten Fällen brauchen Sie keine Deployment-Reihenfolge festzulegen. Verwendet Ihre Applikation Abhängigkeitseinspeisungen und resource-refs zum Bezug auf externe Module. In den meisten Fällen wird das **<initialize-in-order>**-Element nicht benötigt, da der Applikationsserver implizit die korrekte und optimale Weise der Komponentenreihenfolge bestimmen kann.

Angenommen Sie haben eine Applikation, die ein **myBeans.jar** und ein **myApp.war** innerhalb eines **myApp.ear** enthält. Ein Servlet im **myApp.war** verwendet eine **@EJB**-Annotation, um ein Bean aus **myBeans.jar** einzuspeisen. In diesem Fall besitzt der Applikationsserver die entsprechenden Informationen um sicherzustellen, dass die EJB-Komponente verfügbar ist, ehe das Servlet gestartet wird, und Sie müssen das **<initialize-in-order>**-Element nicht benutzen.

Falls dieses Servlet jedoch ältere Remote-Verweise im JNDI-Lookup-Stil verwendet wie nachfolgend für den Zugriff auf das Bean verwendet, so kann es sein, dass Sie die Modulreihenfolge festlegen müssen.

```
init() {
  Context ctx = new InitialContext();
  ctx.lookup("TheBeanInMyBeansModule");
}
```

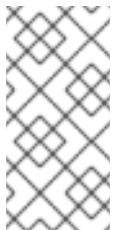
In diesem Fall kann der Server nicht bestimmen, dass sich die EJB-Komponente im **myBeans.jar**

befindet, und Sie müssen erzwingen, dass die Komponenten im **myBeans.jar** vor den Komponenten in **myApp.war** initialisiert und gestartet werden. Um dies zu tun setzen Sie das **<initialize-in-order>**-Element auf **true** und legen die Reihenfolge der **myBeans.jar**- und **myApp.war**-Modules in der **application.xml**-Datei fest.

Nachfolgend sehen Sie ein Beispiel, das das **<initialize-in-order>**-Element zur Steuerung der Deployment-Reihenfolge verwendet. Das **myBeans.jar** wird vor der **myApp.war**-Datei deployt.

```
<application xmlns="http://java.sun.com/xml/ns/javaee"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             version="6"
             xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/application_6.xsd">
  <application-name>myApp</application-name>
  <initialize-in-order>true</initialize-in-order>
  <module>
    <ejb>myBeans.jar</ejb>
  </module>
  <module>
    <web>
      <web-uri>myApp.war</web-uri>
      <context-root>myApp</context-root>
    </web>
  </module>
</application>
```

Das Schema für die **application.xml** finden Sie hier  
[http://java.sun.com/xml/ns/javaee/application\\_6.xsd](http://java.sun.com/xml/ns/javaee/application_6.xsd).



## ANMERKUNG

Die Einstellung des **<initialize-in-order>**-Elements auf **true** verlangsamt das Deployment. Es ist besser, ordnungsgemäße Abhängigkeiten mittels Abhängigkeitseinspeisung oder resource-refs vorzunehmen, da dies dem Container mehr Flexibilität bei der Optimierung von Deployments einräumt.

## jboss-ejb3.xml

Der **jboss-ejb3.xml**-Deployment-Deskriptor ersetzt den **jboss.xml**-Deployment-Deskriptor bei der Außerkraftsetzung und Hinzufügung von Features des von der Java Enterprise Edition (EE) definierten **ejb-jar.xml**-Deployment-Deskriptors. Die neue Datei ist inkompatibel mit **jboss.xml**, und die **jboss.xml** wird jetzt bei Deployments ignoriert.

## login-config.xml

Die **login-config.xml**-Datei wird nicht mehr für die Sicherheitskonfiguration verwendet. Die Sicherheit ist jetzt im **<security-domain>**-Element in der Server-Konfigurationsdatei konfiguriert. Für einen Standalone Server ist dies die **standalone/configuration/standalone.xml**-Datei. Falls Ihr Server in einer Managed Domain läuft, so ist dies die **domain/configuration/domain.xml**-Datei.

[Fehler melden](#)

### 3.1.3.2. jboss-deployment-structure.xml

Bei **jboss-deployment-structure.xml** handelt es sich um einen neuen, optionalen Deployment-Deskriptor für die JBoss EAP 6. Dieser Deployment-Deskriptor bietet Kontrolle über das Klassenladen im Deployment.

Das XML-Schema für diesen Deployment-Deskriptor befindet sich in **EAP\_HOME/docs/schema/jboss-deployment-structure-1\_2.xsd**

[Fehler melden](#)

### 3.1.3.3. Paket-Ressourcen für das neue, modulare Klassenladesystem

#### Zusammenfassung

In früheren Versionen der JBoss EAP wurden alle Ressourcen innerhalb des **WEB-INF/-**Verzeichnisses dem WAR-Klassenpfad hinzugefügt. In der JBoss EAP 6 werden Bausteine der Applikation nur aus den **WEB-INF/classes-** und **WEB-INF/lib-**Verzeichnissen geladen. Werden Applikationsbausteine nicht in den festgelegten Speicherorten verpackt, so kann dies zu **ClassNotFoundException**, **NoClassDefError** oder anderen Runtime Fehlern führen.

Um diese Klassenladefehler aufzulösen, müssen Sie die Struktur Ihrer Applikation archivieren oder ein benutzerdefiniertes Modul definieren.

#### Bearbeitung des Ressourcen-Packagings

Um die Ressourcen nur für die Applikation verfügbar zu machen, müssen Sie die Properties-Dateien, JARs oder andere Bausteine mit dem WAR bündeln, indem Sie sie in das **WEB-INF/classes/-** oder das **WEB-INF/lib/-**Verzeichnis verschieben. Diese Vorgehensweise wird hier [Abschnitt 3.1.3.4, »Speicherort der ResourceBundle Properties ändern«](#) ausführlicher beschrieben.

#### Maßgeschneidertes Modul erstellen

Falls Sie maßgeschneiderte Ressourcen allen auf dem JBoss EAP 6 Server laufenden Applikationen verfügbar machen wollen, so müssen Sie ein maßgeschneidertes Modul erstellen. Diese Vorgehensweise wird hier [Abschnitt 3.1.3.5, »Maßgeschneidertes Modul erstellen«](#) ausführlicher beschrieben.

[Fehler melden](#)

### 3.1.3.4. Speicherort der ResourceBundle Properties ändern

#### Zusammenfassung

In früheren Versionen der JBoss EAP befand sich das **EAP\_HOME/server/SERVER\_NAME/conf/-**Verzeichnis im Klassenpfad und war für die Applikation verfügbar. Um Properties in der JBoss EAP 6 dem Klassenpfad der Applikation verfügbar zu machen, müssen Sie diese innerhalb Ihrer Applikation verpacken.

#### Prozedur 3.2. Speicherort der ResourceBundle Properties ändern

1. Falls Sie ein WAR-Archiv deployen, so müssen Sie diese Properties in den **WEB-INF/classes/-**Ordner des WAR packen.
2. Sollen diese Properties allen Komponenten in einem EAR zugänglich sein, so müssen Sie diese im root eines JARs verpacken und das JAR im **lib/-**Ordner des EAR platzieren.

[Fehler melden](#)

### 3.1.3.5. Maßgeschneidertes Modul erstellen

Die folgende Vorgehensweise beschreibt, wie ein maßgeschneidertes Modul erstellt wird, um Properties-Dateien und andere Ressourcen allen auf dem JBoss EAP Server laufenden Applikationen verfügbar zu machen.

#### Prozedur 3.3. Maßgeschneidertes Modul erstellen

1. Erstellen Sie die **module**/-Verzeichnisstruktur und pflegen Sie die relevanten Daten ein.
  - a. Erstellen Sie eine Verzeichnisstruktur unter dem **EAP\_HOME/module**-Verzeichnis, die die Dateien und JARs enthält. Zum Beispiel:

```
$ cd EAP_HOME/modules/
$ mkdir -p myorg-conf/main/properties
```

- b. Verschieben Sie die Properties-Dateien in das **EAP\_HOME/modules/myorg-conf/main/properties**/-Verzeichnis, das Sie im vorherigen Schritt erstellt haben.
  - c. Erstellen Sie eine **module.xml**-Datei im **EAP\_HOME/modules/myorg-conf/main**/-Verzeichnis, die folgende XML enthält:

```
<module xmlns="urn:jboss:module:1.1" name="myorg-conf">
  <resources>
    <resource-root path="properties"/>
  </resources>
</module>
```

2. Bearbeiten Sie das **ee**-Subsystem in der Server-Konfigurationsdatei. Sie können das JBoss CLI verwenden oder die Datei manuell bearbeiten.
  - o Befolgen Sie diese Schritte zur Bearbeitung der Server-Konfigurationsdatei mittels des JBoss CLI.

- a. Starten Sie den Server und verbinden Sie sich mit dem Management-CLI.

- In Linux geben Sie folgendes in der Befehlszeile ein:

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

- In Windows geben Sie Folgendes in die Befehlszeile ein:

```
C:\>EAP_HOME\bin\jboss-cli.bat --connect
```

Sie sollten die folgende Antwort sehen:

```
Connected to standalone controller at localhost:9999
```

- b. Um das **myorg-conf** <global-modules> Element im **ee** Subsystem zu erstellen, geben Sie folgenden Befehl in die Befehlszeile ein:

```
/subsystem=ee:write-attribute(name=global-modules, value=
[{"name"=>"myorg-conf", "slot"=>"main"}])
```

Sie sollten das folgende Ergebnis sehen:

```
{"outcome" => "success"}
```

- o Führen Sie folgende Schritte aus, um die Server-Konfigurationsdatei manuell zu bearbeiten.
  - a. Stoppen Sie den Server und öffnen Sie die Server-Konfigurationsdatei in einem Texteditor. Für einen Standalone Server ist dies die **EAP\_HOME/standalone/configuration/standalone.xml**-Datei, für eine Managed Domain ist es die **EAP\_HOME/domain/configuration/domain.xml**-Datei.
  - b. Suchen Sie das **ee**-Subsystem und fügen Sie das allgemeine Modul für **myorg-conf** hinzu. Nachfolgend sehen Sie ein Beispiel für das **ee**-Subsystem-Element, das derart bearbeitet wurde, um das **myorg-conf**-Element zu enthalten:

#### Beispiel 3.1. myorg-conf Element

```
<subsystem xmlns="urn:jboss:domain:ee:1.0" >
  <global-modules>
    <module name="myorg-conf" slot="main" />
  </global-modules>
</subsystem>
```

3. Wenn Sie eine Datei namens **my.properties** in den korrekten Modul-Speicherort kopiert haben, so können Sie jetzt Properties-Dateien mittels Code ähnlich dem folgenden laden:

#### Beispiel 3.2. Properties-Datei laden

```
Thread.currentThread().getContextClassLoader().getResource("my.pro
perties");
```

[Fehler melden](#)

### 3.1.4. Protokolländerungen

#### 3.1.4.1. Bearbeitung von Protokollierungsabhängigkeiten

##### Zusammenfassung

Der JBoss LogManager unterstützt Frontends für alle Protokoll-Frameworks, so dass Sie Ihren aktuellen Protokollierungscode beibehalten oder die neue JBoss Protokollierungsinfrastruktur verwenden können. Egal wofür Sie sich entscheiden, Sie werden aufgrund der Änderungen hinsichtlich des modularen Klassenladens Ihre Applikation wahrscheinlich bearbeiten müssen, um die erforderlichen Abhängigkeiten hinzuzufügen.

#### Prozedur 3.4. Aktualisierung des Protokollierungs\_codes der Applikation

1. [Abschnitt 3.1.4.2, »Aktualisierung des Applikationscodes für Protokollierungs-Frameworks von Drittanbietern«](#)
2. [Abschnitt 3.1.4.3, »Bearbeitung des Codes zur Verwendung des neuen JBoss Logging Framework«](#)

[Fehler melden](#)

### 3.1.4.2. Aktualisierung des Applikationscodes für Protokollierungs-Frameworks von Drittanbietern

#### Zusammenfassung

Bei der JBoss EAP 6 werden Logging Abhängigkeiten für übliche Drittanbieter-Frameworks wie Apache Commons Logging, Apache log4j, SLF4J, und Java Logging standardmäßig hinzugefügt. In den meisten Fällen ist die Verwendung von Logging Framework vom JBoss EAP Container zu bevorzugen. Wenn Sie jedoch spezifische Funktionen benötigen, die ein Drittanbieter-Framework bietet, müssen Sie das entsprechende JBoss EAP Modul von Ihrem Deployment ausschließen. Beachten Sie, dass die Serverlogs weiterhin die JBoss EAP Logging Subsystem Konfiguration verwenden, auch wenn Ihr Deployment ein Drittanbieter-Framework benutzt.

Das folgende Verfahren demonstriert, wie Sie das JBoss EAP 6 **org.apache.log4j** Modul von Ihrem Deployment ausschließen. Das erste Verfahren funktioniert in jeder Release von JBoss EAP 6. Das zweite Verfahren kann nur bei JBoss EAP 6.3 oder späteren Versionen angewendet werden.

#### Prozedur 3.5. Konfigurieren Sie die JBoss EAP 6 zur Verwendung einer log4j.properties- oder log4j.xml-Datei

Dieses Verfahren funktioniert für alle Versionen von JBoss EAP 6.



#### ANMERKUNG

Da diese Methode eine log4j-Konfigurationsdatei verwendet, werden Sie die log4j-Protokollierungskonfiguration zur Runtime nicht mehr ändern können.

1. Erstellen Sie eine **jboss-deployment-structure.xml** mit folgendem Inhalt:

```
<jboss-deployment-structure>
  <deployment>
    <!-- Exclusions allow you to prevent the server from
    automatically adding some dependencies -->
    <exclusions>
      <module name="org.apache.log4j" />
    </exclusions>
  </deployment>
</jboss-deployment-structure>
```

2. Platzieren Sie die **jboss-deployment-structure.xml** Datei entweder im **META-INF/** oder im **WEB-INF/** Verzeichnis, wenn Sie ein WAR deployen, oder im **META-INF/** Verzeichnis, wenn Sie ein EAR deployen. Wenn Ihr Deployment abhängige untergeordnete Deployments umfasst, müssen Sie auch die Module für jedes Subdeployment ausschließen.
3. Schließen Sie die **log4j.properties**- oder **log4j.xml**-Datei im **lib/**-Verzeichnis Ihres EAR- oder dem **WEB-INF/classes/**-Verzeichnis Ihres WAR-Deployment ein. Falls Sie es

vorziehen, die Datei in das **lib/**-Verzeichnis Ihres WAR zu legen, müssen Sie den **<resource-root>**-Pfad in der **jboss-deployment-structure.xml**-Datei angeben.

```
<jboss-deployment-structure>
  <deployment>
    <!-- Exclusions allow you to prevent the server from
    automatically adding some dependencies -->
    <exclusions>
      <module name="org.apache.log4j" />
    </exclusions>
    <resources>
      <resource-root path="lib" />
    </resources>
  </deployment>
</jboss-deployment-structure>
```

4. Starten Sie den JBoss EAP 6 Server mit folgendem Laufzeitargument um zu verhindern, dass eine **ClassCastException** in der Konsole erscheint, wenn Sie die Applikation deployen:

```
-Dorg.jboss.as.logging.per-deployment=false
```

5. Deployen Sie Ihre Applikation.

### Prozedur 3.6. Konfigurieren von Logging Abhängigkeiten für JBoss EAP 6.3 oder spätere Versionen

Bei der JBoss EAP 6.3 und späteren Versionen können Sie die neuen **add-logging-api-dependencies** Logging Systemattribute benutzen um Loggingframework-Abhängigkeiten von Drittanbietern auszuschließen. Die folgenden Schritte demonstrieren, wie man dieses Logging-Attribut auf einem JBoss EAP Standalone Server modifiziert.

1. Starten Sie den JBoss EAP 6 Server mit folgendem Laufzeitargument um zu verhindern, dass eine **ClassCastException** in der Konsole erscheint, wenn Sie die Applikation deployen:

```
-Dorg.jboss.as.logging.per-deployment=false
```

2. Öffnen Sie ein Terminal und verbinden Sie sich mit dem Management-CLI.

- o In Linux geben Sie folgendes in der Befehlszeile ein:

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

- o In Windows geben Sie folgendes in der Befehlszeile ein:

```
C:\>EAP_HOME\bin\jboss-cli.bat --connect
```

3. Modifizieren Sie das **add-logging-api-dependencies** Attribut im Logging Subsystem.

Dieses Attribut kontrolliert, ob der Container implizite Logging API Abhängigkeiten zu Ihren Deployments hinzufügt.

- o Wenn **true** gesetzt ist, was der Standardwert ist, sind alle impliziten Logging API Abhängigkeiten hinzugefügt.



- Wenn **false** gesetzt ist, wurden die Abhängigkeiten Ihren Deployments nicht hinzugefügt.

Um die Loggingframework-Abhängigkeiten von Drittanbietern auszuschließen müssen Sie dieses Attribut über folgenden Befehl auf **false** setzen:

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

Dieser Befehl fügt das **<add-logging-api-dependencies>** Element dem **logging** Subsystem der **standalone.xml** Konfigurationsdatei hinzu.

```
<subsystem xmlns="urn:jboss:domain:logging:1.4">
  <add-logging-api-dependencies value="false"/>
  ....
</subsystem>
```

4. Deployen Sie Ihre Applikation.

[Fehler melden](#)

### 3.1.4.3. Bearbeitung des Codes zur Verwendung des neuen JBoss Logging Framework

#### Zusammenfassung

Um das neue Framework zu verwenden, ändern Sie Ihre Importe und Ihren Code wie folgt:

#### Prozedur 3.7. Bearbeitung von Code und Abhängigkeiten zur Verwendung des JBoss Logging Frameworks

1. **Ändern Sie Ihre Importe und Protokollierungscode**

Nachfolgend sehen Sie ein Beispiel für einen Code, der das neue JBoss Logging Framework verwendet:

```
import org.jboss.logging.Level;
import org.jboss.logging.Logger;

private static final Logger logger =
    Logger.getLogger(MyClass.class.toString());

if(logger.isTraceEnabled()) {
    logger.tracef("Starting...", subsystem);
}
```

2. **Fügen Sie die Protokollierungsabhängigkeit hinzu**

Das die JBoss Protokollierungsklassen enthaltende JAR befindet sich im Modul namens **org.jboss.logging**. Ihre **MANIFEST-MF**-Datei sollte wie folgt aussehen:

```
Manifest-Version: 1.0
Dependencies: org.jboss.logging
```

Weitere Informationen dazu, wie Sie die Modulabhängigkeit finden, finden Sie unter [Abschnitt 3.1.2.3, »Aktualisierung der Applikationsabhängigkeiten aufgrund von Klassenladeänderungen«](#) und [Abschnitt 4.2.1, »Fehler- und Problembehebung bei der Migration«](#).

[Fehler melden](#)

### 3.1.5. Änderungen beim Packen der Applikation

#### 3.1.5.1. Modifizierung des Packagings von EARs und WARs

##### Zusammenfassung

Bei Migration Ihrer Applikation kann es sein, dass Sie die Packstruktur Ihres EARs oder WARs aufgrund der Änderung am modularen Klassenladen ändern müssen. Modulabhängigkeiten werden in dieser spezifischen Reihenfolge geladen:

1. Systemabhängigkeiten
2. Benutzerabhängigkeiten
3. Lokale Ressourcen
4. Inter-Deployment Abhängigkeiten

##### Prozedur 3.8. Bearbeitung des Archiv-Packagings

###### 1. Packaging eines WAR

Bei einem WAR handelt es sich um ein einzelnes Modul und alle Klassen im WAR werden mit demselben Klassenlader geladen. Das bedeutet, dass im **WEB-INF/lib**-Verzeichnis gepackte Klassen genauso behandelt werden wie Klassen im **WEB-INF/classes**-Verzeichnis.

###### 2. Packaging eines EAR

Ein EAR besteht aus mehreren Modulen. Das **EAR/lib**-Verzeichnis ist ein einzelnes Modul und jedes WAR oder EJB jar-Unterdeployment innerhalb des EAR ist ein separates Modul. Klassen müssen nicht auf Klassen in anderen Modulen innerhalb des EAR zugreifen, falls nicht explizit Abhängigkeiten definiert wurden. Unterdeployments besitzen eine automatische Abhängigkeit am übergeordneten Modul, das ihnen Zugriff auf die Klassen im **EAR/lib**-Verzeichnis gibt. Allerdings haben Unterdeployments nicht immer eine automatische Abhängigkeit, die ihnen Zugriff auf einander gibt. Dieses Verhalten wird durch Einstellung des **<ear-subdeployments-isolated>**-Element in der **ee**-Subsystemkonfiguration wie folgt gesteuert:

```
<subsystem xmlns="urn:jboss:domain:ee:1.0" >  
  <ear-subdeployments-isolated>false</ear-subdeployments-isolated>  
</subsystem>
```

Dies ist standardmäßig auf "false" eingestellt, wodurch die Unterdeployments die zu anderen Unterdeployments innerhalb des EAR gehörenden Klassen sehen können.

Weitere Informationen zum Laden von Klassen finden Sie im Kapitel *Class Loading and Modules* im *Development Guide* für die JBoss EAP 6 unter [https://access.redhat.com/site/documentation/JBoss\\_Enterprise\\_Application\\_Platform/](https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/).

[Fehler melden](#)

### 3.1.6. Konfigurationsänderungen bei Datenquellen und Ressourcenadaptern

#### 3.1.6.1. Aktualisierung der Applikation aufgrund von Konfigurationsänderungen

Bei der JBoss EAP 5 wurden Dienste und Subsysteme in vielen verschiedenen Dateien konfiguriert. Bei der JBoss EAP 6 erfolgt die Konfiguration nun hauptsächlich in einer Datei. Falls Ihre Applikation eine der folgenden Ressourcen oder einen der folgenden Dienste verwendet, so können Konfigurationsänderungen erforderlich sein.

1. Falls Ihre Applikation eine Datenquelle verwendet, siehe: [Abschnitt 3.1.6.2, »Aktualisierung der Datenquellen-Konfiguration«](#).
2. Falls Ihre Applikation JPA verwendet und derzeit die Hibernate JARs bündelt, so finden Sie Informationen zu Migrationsoptionen hier: [Abschnitt 3.1.6.4, »Konfiguration der Datenquelle für Hibernate oder JPA«](#).
3. Falls Ihre Applikation einen Ressourcenadapter verwendet, siehe: [Abschnitt 3.1.6.5, »Aktualisierung der Ressourcen-Adapter-Konfiguration«](#).
4. Sehen Sie sich die folgenden Informationen zur Konfiguration von Änderungen für grundlegende Sicherheit an: [Abschnitt 3.1.7.1, »Konfiguration der Sicherheitsänderungen der Applikation«](#).

[Fehler melden](#)

### 3.1.6.2. Aktualisierung der Datenquellen-Konfiguration

#### Zusammenfassung

In früheren Versionen von JBoss EAP war die JCA Datenquellen-Konfiguration in einer Datei mit dem Suffix **\*-ds.xml** definiert. Diese Datei wurde dann im **deploy/** Verzeichnis des Servers deployt oder in die Applikation gepackt. Der JDBC Treiber wurde ins **server/lib/** Verzeichnis kopiert oder in das **WEB-INF/lib/** Verzeichnis der Applikation gepackt. Während diese Methode der Datenquellen-Konfiguration für die Entwicklung noch unterstützt wird, wird sie für die Produktion nicht empfohlen, da sie nicht von den JBoss Administrations- und Management-Tools unterstützt wird.

Bei der JBoss EAP 6 wird die Datenquelle in der Serverkonfigurationsdatei konfiguriert. Wenn die JBoss EAP 6 Instanz in einer Managed Domain läuft, wird die Datenquelle in der **domain/configuration/domain.xml**-Datei konfiguriert. Wenn die JBoss EAP Instanz als Standalone Server läuft, wird die Datenquelle in der **standalone/configuration/standalone.xml** file-Datei konfiguriert. Auf diese Weise konfigurierte Datenquellen können mittels der JBoss Management-Interfaces, einschließlich der Web Management-Konsole und des Befehlszeilen-Interface (CLI) verwaltet und gesteuert werden. Diese Tools machen die Verwaltung von Deployments und die Konfiguration mehrerer in einer Managed Domain laufender Server einfach.

Der folgende Abschnitt beschreibt, wie Sie Ihre Datenquellenkonfiguration so bearbeiten, dass diese mit den verfügbaren Management-Tools verwaltet und unterstützt wird.

#### Zu einer für die für JBoss EAP 6 verwaltbaren Datenquellenkonfiguration migrieren

Ein JDBC 4.0 kompatibler Treiber kann als Deployment- oder Kernmodul installiert werden. Ein JDBC 4.0 kompatibler Treiber enthält eine **META-INF/services/java.sql.Driver**-Datei, die den Treiberklassennamen festlegt. Ein nicht mit JDBC 4.0 kompatibler Treiber erfordert zusätzliche Schritte. Informationen dazu, wie Sie einen Treiber JDBC 4.0 kompatibel machen und wie Sie Ihre aktuelle Datenquellenkonfiguration so aktualisieren, dass Sie durch die Web Management-Konsole und das CLI verwaltet werden kann, finden Sie hier [Abschnitt 3.1.6.3, »Installation und Konfiguration des JDBC-Treibers«](#).

Falls Ihre Applikation Hibernate oder JPA verwendet, so sind möglicherweise zusätzliche Änderungen notwendig. Siehe [Abschnitt 3.1.6.4, »Konfiguration der Datenquelle für Hibernate oder JPA«](#) für weitere Informationen.

## Verwendung des IronJacamar Migrationstools zur Konvertierung von Konfigurationsdaten

Sie können das IronJacamar Tool zur Migration von Datenquellen und Ressourcenadapter-Konfigurationen verwenden. Dieses Tool konvertiert die Konfigurationsdateien im **\*-ds.xml**-Stil in das von der JBoss EAP 6 erwartete Format. Weitere Informationen finden Sie unter [Abschnitt 4.1.6, »Verwendung des IronJacamar Migrationstools zur Migration der Datenquellen- und Ressourcenadapterkonfigurationen«](#).

### Migrations-Code, der Remote Datenquellen-Lookups ausführt

In früheren Versionen der JBoss EAP war es möglich ein JNDI Remote Lookup von Datenquellen-Objekten auszuführen, diese Praxis war jedoch aus folgenden Gründen nicht empfohlen.

- Clientenkontrolle einer Serverressource ist unzuverlässig und kann zu undichten Verbindungen führen, wenn der Client abstürzt oder die Verbindung zum Server verliert.
- Die Performance ist sehr langsam, weil alle Datenbankenvorgänge durch einen **MBean** über einen Proxy übermittelt werden.
- Transaktionspropagierung ist nicht unterstützt.

Diese Funktionalität wurde aus JBoss EAP 6 entfernt und Ihnen wird vielleicht ein **NotSerializableException** angezeigt, wenn Sie Ihre Applikation migrieren. Der empfohlene Ansatz ist es, eine EJB zu erstellen um auf die Datenquelle zuzugreifen und dann die EJB remote abzurufen. Weitere Informationen finden Sie im Abschnitt *Remote Invocation Changes* dieses Buches. Zusätzliche Informationen finden Sie im *Development Guide* für JBoss EAP 6 unter [https://access.redhat.com/site/documentation/JBoss\\_Enterprise\\_Application\\_Platform/](https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/).

[Fehler melden](#)

### 3.1.6.3. Installation und Konfiguration des JDBC-Treibers

#### Zusammenfassung

Der JDBC-Treiber kann auf eine der folgenden beiden Arten in den Container installiert werden:

- Als ein Deployment
- Als ein Kernmodul

Die Vor- und Nachteile jeder der Vorgehensweisen finden Sie nachfolgend.

Bei der JBoss EAP 6 wird die Datenquelle in der Serverkonfigurationsdatei konfiguriert. Läuft die JBoss EAP 6 Instanz in einer Managed Domain, so wird die Datenquelle in der **domain/configuration/domain.xml**-Datei konfiguriert. Läuft die JBoss Enterprise Application Platform Instanz als Standalone Server, so wird die Datenquelle in der **standalone/configuration/standalone.xml file**-Datei konfiguriert. Schemareferenzinformationen, die für beide Modi identisch sind, finden Sie im **doc/-**Verzeichnis der JBoss EAP 6 Installation. Wir gehen hier davon aus, dass der Server als Standalone Server läuft und die Datenquelle in der **standalone.xml**-Datei konfiguriert ist.

#### Prozedur 3.9. Installation und Konfiguration des JDBC-Treibers

##### 1. Installation des JDBC-Treibers.

###### a. Installation des JDBC-Treibers als ein Deployment.

Dies ist die empfohlene Vorgehensweise, auf die der Treiber installiert werden sollte. Wenn

der JDBC Treiber als Deployment installiert ist, wird er als normales JAR deployt. Wenn die JBoss EAP 6 Instanz als Standalone Server läuft, so kopieren Sie das JDBC 4.0 kompatible JAR ins **EAP\_HOME/standalone/deployments/** Verzeichnis. Für eine Managed Domain müssen Sie die Management Konsole oder das Management CLI benutzen um das JAR zu den Servergruppen zu deployen.

Nachfolgend sehen Sie ein Beispiel für einen als Deployment an einem Standalone Server installierten MySQL JDBC-Treiber:

```
$cp mysql-connector-java-5.1.15.jar
EAP_HOME/standalone/deployments/
```

Jeder JDBC 4.0 kompatible Treiber wird automatisch erkannt und wird nach Namen und Version im System installiert. Ein JDBC 4.0 kompatibles JAR enthält eine Textdatei namens **META-INF/services/java.sql.Driver**, die den/die Treiberklassenname(n) festlegt. Ist der Treiber nicht JDBC 4.0 kompatibel, so kann er auf eine der folgenden Arten deploybar gemacht werden:

- Erstellen Sie eine **java.sql.Driver**-Datei für das JAR und fügen Sie diese unter dem **META-INF/services/**-Pfad hinzu. Diese Datei sollte den Treiberklassennamen enthalten, zum Beispiel:

```
com.mysql.jdbc.Driver
```

- Erstellen Sie eine **java.sql.Driver** Datei im Deployment Verzeichnis. Für eine als Standalone Server laufende JBoss EAP 6 Instanz sollte die Datei hier sein: **EAP\_HOME/standalone/deployments/META-INF/services/java.sql.Driver**. Wenn der Server in einer Managed Domain ist, müssen Sie die Datei über die Managementkonsole oder das Management CLI deployen.

Die Vorteile dieser Vorgehensweisen sind:

- Dies ist die einfachste Methode, da kein Modul definiert werden muss.
- Läuft der Server in einer Managed Domain, so werden nach dieser Vorgehensweise verwendende Deployments automatisch zu allen Servern in der Domain fortgepflanzt. Dies bedeutet, dass der Administrator das Treiber-JAR nicht manuell distribuieren muss.

Die Nachteile dieser Vorgehensweisen sind:

- Falls der JDBC-Treiber aus mehr als einem JAR besteht, zum Beispiel dem Treiber-JAR plus einem abhängigen Lizenz-JAR oder Lokalisierungs-JAR, so können Sie den Treiber nicht als Deployment installieren. Sie müssen den JDBC-Treiber als ein Kernmodul installieren.
- Falls der Treiber nicht JDBC 4.0 kompatibel ist, so muss eine Datei erstellt werden, die die/den Treiberklassennamen enthält und muss in das JAR importiert werden oder im **deployments/**-Verzeichnis überlagert werden.

#### b. Installation des JDBC-Treibers als Kernmodul.

Um einen JDBC-Treiber als Kernmodul zu installieren, müssen Sie eine Dateipfadstruktur unter dem **EAP\_HOME/modules/**-Verzeichnis erstellen. Diese Struktur enthält das JDBC-Treiber JAR, zusätzliche Anbieter Lizenzen oder Lokalisierungs-JARs und eine **module.xml**-Datei, um das Modul zu definieren.

## ■ Installation des MySQL-Treibers als Kernmodul

- i. Erstellen Sie die Verzeichnisstruktur **EAP\_HOME/modules/com/mysql/main/**
- ii. Erstellen Sie im **main/**-Unterverzeichnis eine **module.xml**-Datei, die die folgende Moduldefinition für den MySQL JDBC-Treiber enthält:

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.0" name="com.mysql">
  <resources>
    <resource-root path="mysql-connector-java-5.1.15.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
  </dependencies>
</module>
```

Der Modulname "com.mysql" stimmt mit der Verzeichnisstruktur für dieses Modul überein. Das **<dependencies>**-Element wird zur Festlegung der Abhängigkeiten von anderen Modulen dieses Moduls verwendet. In diesem Fall (wie auch bei allen anderen JDBC-Datenquellen) hängt dies von den Java JDBC APIs ab, die in einem anderen Modul namens **javax.api** definiert sind. Dieses Modul befindet sich im **modules/system/layers/base/javax/api/main/**-Verzeichnis.



### ANMERKUNG

Stellen Sie sicher, dass sich KEINE Leerstelle am Anfang der **module.xml**-Datei befindet, da Sie sonst die Fehlermeldung "New missing/unsatisfied dependencies" für diesen Treiber erhalten.

- iii. Kopieren Sie das MySQL JDBC-Treiber JAR in das **EAP\_HOME/modules/com/mysql/main/**-Verzeichnis:

```
$ cp mysql-connector-java-5.1.15.jar
EAP_HOME/modules/com/mysql/main/
```

## ■ Installation des IBM DB2 JDBC-Treibers und Lizenz-JAR als Kernmodul.

Dieses Beispiel zeigt Ihnen nur, wie Treiber deployt werden, die neben dem DBC Driver JAR zusätzliche JARs benötigen.

- i. Erstellen Sie die Verzeichnisstruktur **EAP\_HOME/modules/com/ibm/db2/main/**.
- ii. Erstellen Sie im **main/**-Unterverzeichnis eine **module.xml**-Datei, die die folgende Moduldefinition für den IBM DB2 JDBC-Treiber und Lizenz enthält:

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.1" name="com.ibm.db2">
  <resources>
    <resource-root path="db2jcc.jar"/>
    <resource-root path="db2jcc_license_cisuz.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
  </dependencies>
</module>
```

```
<module name="javax.transaction.api"/>
</dependencies>
</module>
```



### ANMERKUNG

Stellen Sie sicher, dass sich KEINE Leerstelle am Anfang der **module.xml**-Datei befindet, da Sie sonst die Fehlermeldung "New missing/unsatisfied dependencies" für diesen Treiber erhalten.

- iii. Kopieren Sie den JDBC-Treiber und das Lizenz-JAR in das **EAP\_HOME/modules/com/ibm/db2/main/-**Verzeichnis.

```
$ cp db2jcc.jar EAP_HOME/modules/com/ibm/db2/main/
$ cp db2jcc_license_cisuz.jar
EAP_HOME/modules/com/ibm/db2/main/
```

Die Vorteile dieser Vorgehensweisen sind:

- Dies ist die einzige Vorgehensweise, die funktioniert, wenn der JDBC-Treiber aus mehr als einem JAR besteht.
- Mit dieser Vorgehensweise können Treiber, die nicht JDBC 4.0 kompatibel sind, installiert werden, ohne dass das Treiber-JAR bearbeitet werden muss oder ein Datei-Overlay erstellt wird.

Die Nachteile dieser Vorgehensweisen sind:

- Es ist schwieriger, ein Modul einzurichten.
- Das Modul muss manuell in jeden in einer Managed Domain laufenden Server kopiert werden.

## 2. Konfiguration der Datenquelle.

### a. Fügen Sie den Datenbanktreiber hinzu.

Fügen Sie das **<driver>**-Element dem **<drivers>**-Element derselben Datei hinzu. Wieder enthält dies einige derselben Datenquelleninformationen, die zuvor in der **\*-ds.xml**-Datei definiert waren.

Bestimmen Sie zuerst, ob das Treiber-JAR JDBC 4.0 kompatibel ist. Ein JAR, das JDBC 4.0 kompatibel ist, enthält eine **META-INF/services/java.sql.Driver**-Datei, die den Treiberklassennamen festlegt. Der Server verwendet diese Datei, um den Namen der Treiberklasse(n) im JAR zu finden. Ein JDBC 4.0 kompatibler Treiber benötigt kein **<driver-class>**-Element, da es bereits im JAR festgelegt ist. Dies ist ein Beispiel des Treiberelements für einen JDBC 4.0 kompatiblen MySQL-Treiber:

```
<driver name="mysql-connector-java-5.1.15.jar"
module="com.mysql"/>
```

Ein Treiber, der nicht JDBC 4.0 kompatibel ist, benötigt ein **<driver-class>**-Attribut zur Identifikation der Treiberklasse, da es keine **META-INF/services/java.sql.Driver**-Datei gibt, die den Treiberklassennamen festlegt. Dies ist ein Beispiel für ein Treiberelement für einen Treiber, der nicht JDBC 4.0 kompatibel ist:

■

```
<driver name="mysql-connector-java-5.1.15.jar"
module="com.mysql">
<driver-class>com.mysql.jdbc.Driver</driver-class></driver>
```

#### b. Erstellen Sie die Datenquelle.

Erstellen Sie ein **<datasource>**-Element im **<datasources>**-Abschnitt der **standalone.xml**-Datei. Diese Datei enthält viele derselben Datenquelleninformationen, die zuvor in der **\*-ds.xml**-Datei definiert waren.



#### WICHTIG

Sie müssen den Server stoppen, ehe Sie die Server Konfigurations-Datei bearbeiten, damit Ihre Änderungen bei einem Server-Neustart als dauerhaft erstellt werden.

Nachfolgend sehen Sie ein Beispiel für ein MySQL-Datenquellenelement in der **standalone.xml**-Datei.

```
<datasource jndi-name="java:/YourDatasourceName" pool-
name="YourDatasourceName">
  <connection-
url>jdbc:mysql://localhost:3306/YourApplicationURL</connection-
url>
  <driver>mysql-connector-java-5.1.15.jar</driver>
  <transaction-isolation>TRANSACTION_READ_COMMITTED</transaction-
isolation>
  <pool>
    <min-pool-size>100</min-pool-size>
    <max-pool-size>200</max-pool-size>
  </pool>
  <security>
    <user-name>USERID</user-name>
    <password>PASSWORD</password>
  </security>
  <statement>
    <prepared-statement-cache-size>100</prepared-statement-cache-
size>
    <share-prepared-statements/>
  </statement>
</datasource>
```

#### 3. Aktualisieren Sie JNDI Referenzen im Applikationscode.

Sie müssen abgelaufene JNDI-Lookup Namen im Applikationsquellencode ersetzen um die neuen, von Ihnen definierten JNDI standardisierten Datenquellen-Namen zu benutzen. Weitere Informationen finden Sie unter: [Abschnitt 3.1.8.4, »Bearbeitung der Applikation zur Befolgung der neuen JNDI-Namespace Regeln«](#).

Sie müssen auch vorhandene **@Resource** Annotationen ersetzen, die auf die Datenquellen zugreifen, um die neuen JNDI-Namen zu benutzen. Zum Beispiel:

```
@Resource(name = "java:/YourDatasourceName").
```



### 3.1.6.4. Konfiguration der Datenquelle für Hibernate oder JPA

Falls Ihre Applikation JPA verwendet und derzeit die Hibernate JARs bündelt, so möchten Sie vielleicht das in der JBoss EAP 6 enthaltene Hibernate verwenden. Um diese Version von Hibernate zu benutzen, müssen Sie das alte Hibernate Bündel aus Ihrer Applikation entfernen.

#### Prozedur 3.10. Entfernen des Hibernate Bündels

1. Entfernen Sie die Hibernate JARs aus den Bibliotheksordnern Ihrer Applikation.
2. Entfernen oder kommentieren Sie das `<hibernate.transaction.manager_lookup_class>`-Element in Ihrer `persistence.xml`-Datei aus, da dieses Element nicht benötigt wird.

[Fehler melden](#)

### 3.1.6.5. Aktualisierung der Ressourcen-Adapter-Konfiguration

#### Zusammenfassung

In früheren Versionen des Applikationsservers war die Ressourcenadapterkonfiguration in einer Datei mit einem Suffix von `*-ds.xml` definiert. Bei der JBoss EAP 6 wird ein Ressourcenadapter in der Server-Konfigurationsdatei definiert. Falls Sie eine Managed Domain betreiben, so ist die Konfigurationsdatei die `EAP_HOME/domain/configuration/domain.xml`-Datei. Falls Sie einen Standalone Server betreiben, so konfigurieren Sie den Ressourcenadapter in der `EAP_HOME/standalone/configuration/standalone.xml`-Datei. Schemareferenzinformation, die für beide Modi identisch ist, finden Sie unter *Schemas* auf der IronJacamar-Website unter: <http://www.ironjacamar.org/documentation.html>.



#### WICHTIG

Sie müssen den Server stoppen, ehe Sie die Server Konfigurations-Datei bearbeiten, damit Ihre Änderungen bei einem Server-Neustart als dauerhaft erstellt werden.

#### Definition des Ressourcenadapters

Die Ressourcenadapter-Deskriptorinformationen sind unter dem folgenden Subsystem-Element in der Serverkonfigurationsdatei definiert:

```
<subsystem xmlns="urn:jboss:domain:resource-adapters:1.1"/>
```

Sie werden einige derselben Informationen verwenden, die zuvor in der Ressourcenadapter `*-ds.xml`-Datei definiert wurden.

Nachfolgend sehen Sie ein Beispiel für ein Ressourcenadapter-Element in der Serverkonfigurationsdatei:

```
<resource-adapters>
  <resource-adapter>
    <archive>multiple-full.rar</archive>
    <config-property name="Name">ResourceAdapterValue</config-property>
    <transaction-support>NoTransaction</transaction-support>
    <connection-definitions>
      <connection-definition
        class-
```

```

name="org.jboss.jca.test.deployers.spec.rars.multiple.MultipleManagedConne
ctionFactory1"
    enabled="true" jndi-name="java:/eis/MultipleConnectionFactory1"
    pool-name="MultipleConnectionFactory1">
    <config-property name="Name">MultipleConnectionFactory1Value</config-
property>
    </connection-definition>
    <connection-definition
    class-
name="org.jboss.jca.test.deployers.spec.rars.multiple.MultipleManagedConne
ctionFactory2"
    enabled="true" jndi-name="java:/eis/MultipleConnectionFactory2"
    pool-name="MultipleConnectionFactory2">
    <config-property name="Name">MultipleConnectionFactory2Value</config-
property>
    </connection-definition>
</connection-definitions>
<admin-objects>
    <admin-object
    class-
name="org.jboss.jca.test.deployers.spec.rars.multiple.MultipleAdminObject1
Impl"
    jndi-name="java:/eis/MultipleAdminObject1">
    <config-property name="Name">MultipleAdminObject1Value</config-
property>
    </admin-object>
    <admin-object class-
name="org.jboss.jca.test.deployers.spec.rars.multiple.MultipleAdminObject2
Impl"
    jndi-name="java:/eis/MultipleAdminObject2">
    <config-property name="Name">MultipleAdminObject2Value</config-
property>
    </admin-object>
</admin-objects>
</resource-adapter>
</resource-adapters>

```

[Fehler melden](#)

### 3.1.7. Sicherheitsänderungen

#### 3.1.7.1. Konfiguration der Sicherheitsänderungen der Applikation

##### Konfiguration der Sicherheit für einfache Authentifizierung

In früheren Versionen der JBoss EAP befanden sich properties-Dateien im **EAP\_HOME/server/SERVER\_NAME/conf/**-Verzeichnis im Klassenpfad und wurden leicht vom **UsersRolesLoginModule** aufgefunden. Bei der JBoss EAP 6 hat sich die Verzeichnisstruktur geändert. Properties-Dateien müssen jetzt innerhalb der Applikation verpackt sein, um im Klassen-Pfad verfügbar zu sein.



#### WICHTIG

Sie müssen den Server stoppen, ehe Sie die Server Konfigurations-Datei bearbeiten, damit Ihre Änderungen bei einem Server-Neustart als dauerhaft erstellt werden.

Für die Konfiguration der Sicherheit für einfache Authentifizierung fügen Sie eine neue Sicherheits-Domain unter **security-domains** zur **standalone/configuration/standalone.xml** oder zur **domain/configuration/domain.xml** Server Konfigurations-Datei hinzu:

```
<security-domain name="example">
  <authentication>
    <login-module code="UsersRoles" flag="required">
      <module-option name="usersProperties"
        value="${jboss.server.config.dir}/example-
users.properties"/>
      <module-option name="rolesProperties"
        value="${jboss.server.config.dir}/example-
roles.properties"/>
    </login-module>
  </authentication>
</security-domain>
```

Läuft die JBoss EAP 6 Instanz als ein Standalone Server, so bezieht sich

**`${jboss.server.config.dir}`** auf das **`EAP_HOME/standalone/configuration/`** Verzeichnis.

Läuft die Instanz in einer Managed Domain, so bezieht sich **`${jboss.server.config.dir}`** auf das **`EAP_HOME/domain/configuration/`** Verzeichnis.

### Bearbeitung der Namen von Sicherheits-Domains

Bei der JBoss EAP 6 verwenden Sicherheits-Domains nicht mehr das Präfix **`java:/jaas/`** in ihrem Namen.

- Für Webanwendungen müssen Sie dieses Präfix aus den Sicherheits-Domain Konfigurationen in **`jboss-web.xml`** entfernen.
- Für Enterprise Anwendungen müssen Sie dieses Präfix aus den Sicherheits-Domain Konfigurationen in **`jboss-ejb3.xml`** entfernen. Diese Datei hat die **`jboss.xml`** in der JBoss EAP 6 ersetzt.

### Fehler melden

### 3.1.7.2. Aktualisieren von Applikationen, die PicketLink STS und Web Services benutzen

#### Zusammenfassung

Wenn Ihre JBoss EAP 6.1 Applikation PicketLink STS und Web Services benutzt, müssen Sie vielleicht Änderungen vornehmen, wenn Sie zu JBoss EAP 6.2 oder späteren Versionen migrieren. Eine Fehlerbehebung, die auf JBoss EAP angewendet wird um [CVE-2013-2133](#) zu adressieren, erzwingt Autorisierungsüberprüfungen durch den Container bevor an EJB3-based WS Endpunkte angefügte JAXWS Handler ausgeführt werden können. Daraus folgt, dass manche PicketLink STS Funktionalitäten betroffen sein können, weil der PicketLink **`SAML2Handler`** ein Sicherheitsprinzipal erstellt, das später in diesem Prozess verwendet werden soll. Sie sehen vielleicht eine **`NullPointerException`** im Serverlog, da das Prinzipal **`NULL`** ist, wenn der **`HandlerAuthInterceptor`** auf den **`SAML2Handler`** zugreift. Sie müssen diese Sicherheitsüberprüfung deaktivieren um dieses Problem zu beheben.

#### Prozedur 3.11. Zusätzliche Autorisierungsüberprüfungen deaktivieren

- Sie können die zusätzlichen Autorisierungsüberprüfungen und die vorhandenen PicketLink Deployments über eine der folgenden Methoden deaktivieren.
  - **Einrichten einer System-weiten Property**

Sie können zusätzliche Autorisierungsüberprüfungen auf Serverebene deaktivieren, indem Sie den **org.jboss.ws.cxf.disableHandlerAuthChecks** System-Property Wert auf **true** setzen. Diese Methode betrifft jedes Deployment, das auf dem Server gemacht wurde.

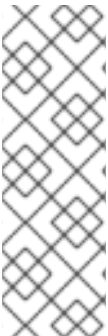
Informationen darüber, wie Sie eine System-Property einrichten, finden Sie unter dem Abschnitt *Configure System Properties Using the Management CLI* im *Administration and Configuration Guide* für JBoss EAP.

- o **Erstellen einer Property in der Web Services Deskriptor Datei des Deployments**

Sie können zusätzliche Autorisierungsüberprüfungen auf Deploymentebene deaktivieren, indem Sie den **org.jboss.ws.cxf.disableHandlerAuthChecks** Property Wert auf **true** in der **jboss-webservices.xml** Datei setzen. Diese Methode wirkt sich nur auf das spezifische Deployment aus.

- a. Erstellen Sie eine **jboss-webservices.xml** Datei im **META-INF/** Verzeichnis des Deployments, für das Sie die zusätzlichen Autorisierungsüberprüfungen deaktivieren möchten.
- b. Fügen Sie folgenden Inhalt hinzu:

```
<?xml version="1.1" encoding="UTF-8"?>
<webservices xmlns="http://www.jboss.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  version="1.2"
  xsi:schemaLocation="http://www.jboss.com/xml/ns/javaee">
  <property>
    <name>org.jboss.ws.cxf.disableHandlerAuthChecks</name>
    <value>true</value>
  </property>
</webservices>
```



## ANMERKUNG

Das Aktivieren der **org.jboss.ws.cxf.disableHandlerAuthChecks** Property rendert ein System, das anfällig ist auf [CVE-2013-2133](#). Wenn die Applikation die Anwendung von bei EJB Methoden deklarierten Sicherheitseinschränkungen erwartet, und diese nicht unabhängig vom JAX-WS Handler anwendet, dann sollte die Property nicht aktiviert werden. Die Property sollte nur für Abwärtskompatibilitäts-Zwecke verwendet werden, wenn sie zur Vermeidung eines Zusammenbruchs der Applikation benötigt wird.

[Fehler melden](#)

## 3.1.8. JNDI-Änderungen

### 3.1.8.1. Aktualisierung der JNDI Namespace-Namen der Applikation

#### Zusammenfassung

Mit EJB 3.1 wurden ein allgemeiner, standardisierter JNDI-Namespace und eine Reihe damit verbundener Namespaces eingeführt, die zu den verschiedenen Bereichen einer Java EE Applikation mappen. Die drei JNDI-Namespaces, die für portierbare EJB-Namen verwendet werden, sind **java:global**, **java:module** und **java:app**. Applikationen mit EJB, die JNDI verwenden, müssen so geändert werden, dass Sie der neuen, standardisierten JNDI-Namespace Konvention folgen.

## Prozedur 3.12. JNDI-Lookups bearbeiten

1. Erfahren Sie mehr über [Abschnitt 3.1.8.2, »Portierbare EJB JNDI-Namen«](#)
2. [Abschnitt 3.1.8.3, »Übersicht über die JNDI-Namespace Regeln«](#)
3. [Abschnitt 3.1.8.4, »Bearbeitung der Applikation zur Befolgung der neuen JNDI-Namespace Regeln«](#)

## Beispiele für JNDI-Mappings

Beispiele von JNDI-Namespace in früheren Releases und wie diese in der JBoss EAP 6 spezifiziert sind, finden Sie hier: [Abschnitt 3.1.8.5, »Beispiele von JNDI-Namespace in früheren Releases und wie diese in der JBoss EAP 6 spezifiziert sind«](#)

[Fehler melden](#)

## 3.1.8.2. Portierbare EJB JNDI-Namen

### Zusammenfassung

Die Java EE 6 Spezifikation definiert vier logische Namespaces, jeden mit seinem eigenen Bereich, aber portierbare EJB Names werden nur an drei von ihnen gebunden. Die folgende Tabelle zeigt, wann und wie jeder Namespace verwendet wird.

**Tabelle 3.1. Portierbare JNDI-Namespace**

JNDI-Namespace	Beschreibung
java:global	<p>Namen in diesem Namespace werden von allen in einer Applikationsserverinstanz deployten Applikationen geteilt. Verwenden Sie Namen um auf demselben Server deployte Archive von EJBs zu finden.</p> <p>Im Folgenden sehen Sie ein Beispiel für ein java:global namespace:  <b>java:global/jboss-seam-booking/jboss-seam-booking-jar/HotelBookingAction</b></p>
java:module	<p>Namen in diesem Namespace werden von allen Komponenten in einem Modul geteilt, zum Beispiel allen Enterprise Beans in einem einzelnen EJB-Modul oder allen Komponenten in einem Web-Modul.</p> <p>Im Folgenden sehen Sie ein Beispiel für ein java:module namespace:  <b>java:module/HotelBookingAction!org.jboss.seam.example.booking.HotelBooking</b></p>
java:app	<p>Namen in diesem Namespace werden von allen Komponenten in allen Modulen in einer einzelnen Applikation geteilt. Ein WAR und eine EJB Jar-Datei in derselben EAR-Datei hätten Zugriff auf die Ressourcen im java:app namespace.</p> <p>Im Folgenden sehen Sie ein Beispiel für ein java:app namespace:  <b>java:app/jboss-seam-booking-jar/HotelBookingAction</b></p>

Weitere Informationen zu JNDI-Namensgebungungskontexten finden Sie in Abschnitt EE.5.2.2, "Application Component Environment Namespaces" in der "JSR 316: Java™ Platform, Enterprise Edition (Java EE) Specification, v6". Sie können die Spezifikation hier herunterladen:

<http://jcp.org/en/jsr/detail?id=316>

[Fehler melden](#)

### 3.1.8.3. Übersicht über die JNDI-Namespace Regeln

#### Zusammenfassung

Die JBoss EAP 6 wurde hinsichtlich der JNDI Namespace-Namen verbessert, um berechenbare und konsistente Regeln für jeden im Applikationsserver gebundenen Namen zu liefern sowie um zukünftige Kompatibilitätsprobleme zu vermeiden. Dies bedeutet, dass Sie Probleme mit den aktuellen Namespaces in Ihrer Applikation haben könnten, falls diese die neuen Regeln nicht berücksichtigen.

#### Namespaces sollten folgende Regeln befolgen:

1. Unqualifizierte, relative Namen wie **DefaultDS** oder **jdbc/DefaultDS** sollten je nach Kontext relativ zu **java:comp/env**, **java:module/env** oder **java:jboss/env** qualifiziert werden.
2. Unqualifizierte **absolute** Namen wie **/jdbc/DefaultDS** sollten relativ zu einem **java:jboss/root** Namen qualifiziert werden.
3. Qualifizierte **absolute** Namen wie **java:/jdbc/DefaultDS** sollten auf dieselbe Weise wie unqualifizierte **absolute** Namen oben qualifiziert werden.
4. Der spezielle **java:jboss**-Namespace wird über die gesamte AS Serverinstanz hinweg geteilt.
5. Jeder **relative** Name mit einem **java:-**Präfix muss in einem der fünf Namespaces sein: **comp**, **module**, **app**, **global** oder dem proprietären **jboss**. Jeder mit **java:xxx** beginnende Name, bei dem xxx nicht mit einem der fünf obigen übereinstimmt, führt zu einem "invalid name"-Fehler (ungültiger Name).

[Fehler melden](#)

### 3.1.8.4. Bearbeitung der Applikation zur Befolgung der neuen JNDI-Namespace Regeln

- Nachfolgend sehen Sie ein Beispiel eines JNDI-Lookups in JBoss EAP 5.1. Dieser Code befindet sich in der Regel in einer Initialisierungsmethode.

```
private ProductManager productManager;
try {
    context = new InitialContext();
    productManager = (ProductManager)
context.lookup("OrderManagerApp/ProductManagerBean/local");
} catch(Exception lookupError) {
    throw new ServletException("Unable to find the ProductManager
bean", lookupError);
}
```

Beachten Sie, dass der Lookup-Name **OrderManagerApp/ProductManagerBean/local** lautet.

- Das nachfolgende Beispiel zeigt, wie derselbe Lookup in der JBoss EAP 6 mittels Abhängigkeitseinspeisung codiert würde.

```
@EJB(lookup="java:app/OrderManagerEJB/ProductManagerBean!services.ej
b.ProductManager")
private ProductManager productManager;
```

Die Lookup-Werte sind jetzt als Mitgliedervariablen definiert und verwenden den neuen, portierbaren **java:app** JNDI-Namespace Namen

**java:app/OrderManagerEJB/ProductManagerBean!services.ejb.ProductManager**.

- Falls Sie die Abhängigkeitseinspeisung lieber nicht verwenden möchten, so können Sie weiterhin wie oben InitialContext erstellen und den Lookup bzw. die Suche so bearbeiten, dass der neue JNDI Namespace Name verwendet wird.

```
private ProductManager productManager;
try {
    context = new InitialContext();
    productManager = (ProductManager)
context.lookup("java:app/OrderManagerEJB/ProductManagerBean!services
.ejb.ProductManager");
} catch(Exception lookupError) {
    throw new ServletException("Unable to find the ProductManager
bean", lookupError);
}
```

[Fehler melden](#)

### 3.1.8.5. Beispiele von JNDI-Namespace in früheren Releases und wie diese in der JBoss EAP 6 spezifiziert sind

**Tabelle 3.2. JNDI-Namespace Mapping Tabelle**

Namespace in JBoss EAP 5.x	Namespace in der JBoss EAP 6	Anmerkungen
OrderManagerApp/ProductManagerBean/local	java:module/ProductManagerBean!services.ejb.ProductManager	Java EE 6 Standardbindung. Begrenzt durch das aktuelle Modul und nur innerhalb desselben Moduls zugänglich.
OrderManagerApp/ProductManagerBean/local	java:app/OrderManagerEJB/ProductManagerBean!services.ejb.ProductManager	Java EE 6 Standardbindung. Begrenzt durch die aktuelle Applikation und nur innerhalb derselben Applikation zugänglich.
OrderManagerApp/ProductManagerBean/local	java:global/OrderManagerApp/OrderManagerEJB/ProductManagerBean!services.ejb.ProductManager	Java EE 6 Standardbindung. Begrenzt durch den Applikationsserver und allgemein zugänglich.



Namespace in JBoss EAP 5.x	Namespace in der JBoss EAP 6	Anmerkungen
java:comp/UserTransaction	java:comp/UserTransaction	Der Namespace wird durch die aktuelle Komponente begrenzt. Nicht zugänglich für Threads die kein Java EE 6 sind, z.B. direkt von Ihrer Applikation erstellte Threads.
java:comp/UserTransaction	java:jboss/UserTransaction	Allgemein zugänglich, verwenden Sie dies, falls java:comp/UserTransaction nicht verfügbar ist
java:/TransactionManager	java:jboss/TransactionManager	
java:/TransactionSynchronizationRegistry	java:jboss/TransactionSynchronizationRegistry	

[Fehler melden](#)

### 3.1.9. HTTP/HTTPS/AJP Konnektorattribute zuordnen

#### 3.1.9.1. Zuordnen der HTTP/HTTPS/AJP Connector Attribute

Die folgende Tabelle zeigt, wie man HTTP, HTTPS und AJP Connector Attribute aus früheren Releases den neuen Attributen in Red Hat JBoss Enterprise Application Platform 6 zuordnet.

**Tabelle 3.3. Zuordnen von Connector Attributen**

Attributnamen in der früheren Release	Entsprechung in JBoss EAP 6	Details
maxThreads	max-connections	<p>Dieses Attribut misst den JBossWeb level thread/connection Pool. Es ist auf den Konnektoren im Web Subsystem eingerichtet. Der Standard ist 512 pro CPU Kern.</p> <pre>&lt;connector name="http" protocol="HTTP/1.1" scheme="http" socket-binding="http" enabled="true" max-connections="200" /&gt;</pre>
minSpareThreads  maxSpareThreads	N/A	<p>Es gibt kein Äquivalent für <b>minSpareThreads</b> oder <b>maxSpareThreads</b>, da kaum Anreiz zur Freigabe von Threads besteht. Wenn Sie einen Executor für den Konnektor benutzen anstatt des Standard Worker Thread Pools, wäre das Nächstliegende die <b>core-threads</b> Größe und <b>keepalive-time</b> Attribute.</p>



Attributnamen in der früheren Release	Entsprechung in JBoss EAP 6	Details
proxyName  proxyPort	proxy-name  proxy-port	<p>Dieses Attribut ist auf dem <b>connector</b> im <b>web</b> Subsystem eingerichtet.</p> <pre>&lt;connector name="http" protocol="HTTP/1.1" scheme="http" socket-binding="http" enabled="true" proxy-name="proxy.com" proxy- port="80"/&gt;</pre>
redirectPort	redirect-port	<p>Dieses Attribut ist auf dem <b>connector</b> im <b>web</b> Subsystem eingerichtet.</p> <pre>connector name="http" protocol="HTTP/1.1" scheme="https" secure="true" socket- binding="http"         redirect-port="8443" proxy- name="loadbalancer.hostname.com" proxy- port="443"</pre>
enableLookups	enable-lookups	<p>Dieses Attribut ist auf dem <b>connector</b> im <b>web</b> Subsystem eingerichtet.</p> <pre>&lt;connector name="http" protocol="HTTP/1.1" scheme="http" socket-binding="http" enable- lookups="true"/&gt;</pre>
MaxHttpHeaderSize	System Property	<p>Dieses Attribut wird unter Verwendung der System Properties eingerichtet. Der Standardwert ist 8 KB.</p> <pre>&lt;system-properties&gt;   &lt;property name="org.apache.coyote.http11.Http11Protocol. MAX_HEADER_SIZE" value="8192"/&gt; &lt;/system-properties&gt;</pre>
maxKeepAliveRequests	System Property	<p>Dieses Attribut wird unter Verwendung der System Properties eingerichtet.</p> <pre>&lt;system-properties&gt;   &lt;property name="org.apache.coyote.http11.Http11Protocol. MAX_KEEP_ALIVE_REQUESTS" value="1"/&gt; &lt;/system-properties&gt;</pre>

Attributnamen in der früheren Release	Entsprechung in JBoss EAP 6	Details
connectionTimeout	System Property	<p>Dieses Attribut wird unter Verwendung der System Properties eingerichtet. Folgende Konfiguration setzt das AJP connectionTimeout auf 600000 Millisekunden (10 Minuten) und das HTTP connectionTimeout auf 120000 Millisekunden (2 Minuten).</p> <pre> &lt;system-properties&gt;   &lt;!-- connectionTimeout for AJP connector.   Default value is "-1" (no timeout). --&gt;   &lt;property     name="org.apache.coyote.ajp.DEFAULT_CONNECTION_TIMEOUT" value="600000"/&gt;   &lt;!-- connectionTimeout for HTTP   connector. Default value is "60000". --&gt;   &lt;property     name="org.apache.coyote.http11.DEFAULT_CONNECTION_TIMEOUT" value="120000"/&gt; &lt;/system-properties&gt; </pre>
Kompression	System Property	<p>Dieses Attribut aktiviert Kompression. Sie können den Inhaltstyp bestimmen, der als <b>text/html</b>, <b>text/xml</b>, <b>text/plain</b> vorgegeben ist. Sie können auch eine minimale Größe von zu komprimierendem Inhalt angeben, was als <b>2048</b> Bytes vorgegeben ist. Kompression wird über die System Properties eingerichtet.</p> <pre> &lt;system-properties&gt;   &lt;property     name="org.apache.coyote.http11.Http11Protocol.COMPRESSION" value="on"/&gt;   &lt;property     name="org.apache.coyote.http11.Http11Protocol.COMPRESSION_MIN_SIZE" value="4096"/&gt;   &lt;property     name="org.apache.coyote.http11.Http11Protocol.COMPRESSION_MIME_TYPES"     value="text/javascript,text/css,text/html"/&gt; &lt;/system-properties&gt; </pre>
URIEncoding	System Property	<p>Dieses Attribut wird unter Verwendung der System Properties eingerichtet.</p> <pre> &lt;system-properties&gt;   &lt;property     name="org.apache.catalina.connector.URI_ENCODING" value="UTF-8"/&gt; &lt;/system-properties&gt; </pre>

Attributnamen in der früheren Release	Entsprechung in JBoss EAP 6	Details
useBodyEncodingForURI	System Property	<p>Dieses Attribut wird unter Verwendung der System Properties eingerichtet.</p> <pre>&lt;system-properties&gt;   &lt;property     name="org.apache.catalina.connector.USE_BODY_ENCODING_FOR_QUERY_STRING" value="true"/&gt; &lt;/system-properties&gt;</pre>
server	System Property	<p>Dieses Attribut wird unter Verwendung der System Properties eingerichtet.</p> <pre>&lt;system-properties&gt;   &lt;property     name="org.apache.coyote.http11.Http11Protocol.SERVER" value="NewServerHeader"/&gt; &lt;/system-properties&gt;</pre>
allowTrace	System Property	<p>Dieses Attribut wird unter Verwendung der System Properties eingerichtet.</p> <pre>&lt;system-properties&gt;   &lt;property     name="org.apache.catalina.connector.ALLOW_TRACE " value="true"/&gt; &lt;/system-properties&gt;</pre>
xpoweredby	System Property	<p>Dieses Attribut wird unter Verwendung der System Properties eingerichtet.</p> <pre>&lt;system-properties&gt;   &lt;property     name="org.apache.catalina.connector.X_POWERED_BY " value="true"/&gt; &lt;/system-properties&gt;</pre>
keepAliveTimeout	N/A	<p>Vor der JBoss EAP 6.4 gab es keinen äquivalenten Parameter in JBoss EAP 6. Intern wurde es auf den Standardwert <b>connectionTimeout</b> gesetzt.</p> <p>Bei der JBoss EAP 6.4, einer neuen System Property, wurde <b>org.apache.coyote.http11.DEFAULT_KEEP_ALIVE_TIMEOUT</b> eingeführt um das keepAliveTimeout zu kontrollieren. Das - <b>Dorg.apache.coyote.http11.DEFAULT_KEEP_ALIVE_TIMEOUT</b> Argument ist betroffen, wenn es mit dem - <b>Dorg.apache.coyote.http11.DEFAULT_DISABLE_UPLOAD_TIMEOUT=true</b> Argument verwendet wird.</p>

Attributnamen in der früheren Release	Entsprechung in JBoss EAP 6	Details
disableUploadTimeout  connectionUploadTimeout	N/A	Derzeit gibt es keine äquivalenten Parameter in JBoss EAP 6. Das <b>disableUploadTimeout</b> ist standardmäßig wahr und das <b>connectionUploadTimeout</b> benutzt intern den <b>connectionTimeout</b> Wert.
packetSize	System Property	Dieses Attribut wurde über die System Properties eingerichtet. Die folgende Konfiguration setzt die AJP packetSize auf <b>20000</b> <pre>&lt;system-properties&gt;   &lt;property     name="org.apache.coyote.ajp.MAX_PACKET_SIZE"     value="20000"/&gt; &lt;/system-properties&gt;</pre>
maxPostSize  maxSavePostSize	max-post-size  max-save-post-size	Der Wert <b>0</b> bedeutet unlimitiert. Beachten Sie, dass dieser Parameter die Datengröße nur dann limitieren kann, wenn <b>Content-Type application/x-www-form-urlencoded</b> ist. Weitere Informationen finden Sie unter dieser Lösung im Kundenportal: <a href="#">How to limit data size of HTTP POST method from a client to JBoss</a>
tomcatAuthentication	System Property	Abhängig von der Version von JBoss EAP 6 wird dieses Attribut unter Verwendung der System Property <b>org.apache.coyote.ajp.AprProcessor.TOMCATAUTHENTICATION</b> oder <b>org.apache.coyote.ajp.DEFAULT_TOMCAT_AUTHENTICATION</b> eingerichtet. Ein Patch oder Upgrade ist für alle Versionen des Servers erforderlich.  Bei der JBoss EAP 6.0.1 wird tomcatAuthentication unter Verwendung der folgenden Property konfiguriert. <pre>&lt;system-properties&gt;   &lt;property     name="org.apache.coyote.ajp.AprProcessor.TOMCATAUTHENTICATION"     value="false"/&gt; &lt;/system-properties&gt;</pre> Bei der JBoss EAP 6.1 und späteren Versionen wird tomcatAuthentication folgendermaßen konfiguriert: <pre>&lt;system-properties&gt;   &lt;property     name="org.apache.coyote.ajp.DEFAULT_TOMCAT_AUTHENTICATION"     value="false"/&gt; &lt;/system-properties&gt;</pre> Weitere Informationen finden Sie unter dieser Lösung im Kundenportal: <a href="#">How to configure tomcatAuthentication in JBoss EAP 6</a>

Weitere Informationen über Konnektor Attribute finden Sie unter dieser Lösung im Kundenportal:  
[Equivalent HTTP/HTTPS/AJP connector attributes mapping between JBoss EAP 5.x and JBoss EAP 6.x](#)

[Fehler melden](#)

## 3.2. VON IHRER APPLIKATIONSARCHITEKTUR UND -KOMPONENTEN ABHÄNGIGE ÄNDERUNGEN

### 3.2.1. Übersicht der von Ihrer Applikationsarchitektur und -komponenten abhängigen Änderungen

Falls Ihre Applikation eine der folgenden Technologien oder Komponenten verwendet, so müssen Sie möglicherweise Änderungen an Ihrer Applikation vornehmen, wenn Sie zur JBoss EAP 6 migrieren.

#### Hibernate und JPA

Falls Ihre Applikation Hibernate oder JPA verwendet, so benötigt Ihre Applikationen möglicherweise einige Modifikationen. Weitere Informationen finden Sie unter: [Abschnitt 3.2.2.1, »Aktualisierung von Applikationen, die Hibernate und/oder JPA verwenden«](#).

#### REST

Falls Ihre Applikation JAX-RS verwendet, so sollten Sie wissen, dass die JBoss EAP 6 automatisch RESTEasy einstellt, so dass Sie es nicht mehr selbst konfigurieren müssen. Weitere Informationen finden Sie unter: [Abschnitt 3.2.5.1, »Konfiguration der JAX-RS- und RESTEasy-Änderungen«](#)

#### LDAP

Der LDAP-Sicherheitsbereich wird bei der JBoss EAP 6 anders konfiguriert. Falls Ihre Applikation LDAP verwendet, so finden Sie in folgendem Topic weitere Informationen: [Abschnitt 3.2.6.1, »Konfiguration der Änderungen des LDAP-Sicherheitsbereichs «](#).

#### Nachrichtenvermittlung

JBoss Messaging ist in der JBoss EAP 6 nicht mehr enthalten. Falls Ihre Applikation JBoss Messaging als Messaging-Provider verwendet, so müssen Sie den JBoss Messaging Code durch HornetQ ersetzen. Das folgende Topic beschreibt die Vorgehensweise: [Abschnitt 3.2.7.4, »Migration Ihrer Applikation zur Verwendung von HornetQ als dem JMS-Provider«](#).

#### Clustering

Die Weise, auf die Clustering aktiviert wird, hat sich bei der JBoss EAP 6 geändert. Weitere Informationen finden Sie hier: [Abschnitt 3.2.8.1, »Durchführung von Änderungen an Ihrer Applikation für Clustering«](#).

#### Deployment im Service-Stil

Obwohl die JBoss EAP 6 Deskriptoren im Service-Stil nicht mehr verwendet, unterstützt der Container diese Deployments im Service-Stil wo möglich ohne Änderungen. Deployment-Informationen finden Sie unter: [Abschnitt 3.2.9.1, »Aktualisierung von Applikationen, die Deployments im Dienst-Stil verwenden«](#)

#### Remote-Aufruf

Falls Ihre Applikation Remote-Aufrufe tätigt, so können Sie nach wie vor JNDI zum Auffinden eines Proxy für Ihr Bean verwenden und Aufrufe an diesem wiedergegebenen Proxy durchführen. Weitere Informationen zur erforderlichen Syntax und Namespace-Änderungen finden Sie unter: [Abschnitt](#)

[3.2.10.1, »Migration von JBoss EAP 5 deployten Applikationen, die Remote-Aufrufe zur JBoss EAP 6 tätigen«](#).

## Seam 2.2

Falls Ihre Applikation Seam 2.2 verwendet, so finden Sie Informationen zu den notwendigen Änderungen hier: [Abschnitt 3.2.13.1, »Migration der Seam 2.2 Archive zur JBoss EAP 6«](#).

## Spring

Falls Ihre Applikation Spring verwendet, siehe: [Abschnitt 3.2.14.1, »Migration von Spring Applikationen«](#).

## Andere Änderungen, die Ihre Migration beeinflussen können

Hinsichtlich weiterer Änderungen in der JBoss EAP 6, die Ihre Applikation beeinflussen können, siehe: [Abschnitt 3.2.15.1, »Machen Sie sich mit anderen Änderungen vertraut, die Einfluss auf Ihre Migration haben können«](#).

[Fehler melden](#)

## 3.2.2. Hibernate- und JPA-Änderungen

### 3.2.2.1. Aktualisierung von Applikationen, die Hibernate und/oder JPA verwenden

#### Zusammenfassung

Falls Ihre Applikation Hibernate oder JPA verwendet, so lesen Sie bitte die folgenden Abschnitte und nehmen Sie die notwendigen Änderungen an Ihrer Applikation vor, wenn Sie zur JBoss EAP 6 migrieren.

- [Abschnitt 3.2.2.2, »Konfiguration der Änderungen bei Applikationen, die Hibernate und JPA verwenden«](#)
- [Abschnitt 3.2.2.4, »Aktualisieren Sie Ihre Hibernate 3 Applikation, damit sie Hibernate 4 benutzt«](#)
- [Abschnitt 3.2.2.9, »Aktualisieren Sie Ihre Applikation, damit sie mit der JPA 2.0 Spezifikation konform ist«](#)
- [Abschnitt 3.2.2.10, »Ersetzen des Caches der zweiten Ebene von JPA/Hibernate durch Infinispan«](#)
- [Abschnitt 3.2.2.12, »Migration zu Hibernate Validator 4«](#)

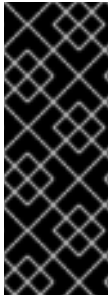
[Fehler melden](#)

### 3.2.2.2. Konfiguration der Änderungen bei Applikationen, die Hibernate und JPA verwenden

#### Zusammenfassung

Falls Ihre Ihre Applikation eine `persistence.xml`-Datei enthält oder der Code die Annotationen `@PersistenceContext` oder `@PersistenceUnit` verwendet, so spürt die JBoss EAP 6 dies während des Deployments auf und geht davon aus, dass die Applikation JPA verwendet. Sie fügt dem Klassenpfad Ihrer Applikation implizit Hibernate 4 sowie ein paar andere Abhängigkeiten hinzu.

Falls Ihre Applikation derzeit Hibernate 3 Bibliotheken verwendet, so können Sie in den meisten Fällen erfolgreich auf Hibernate 4 umstellen. Falls Sie jedoch **ClassNotFoundExceptions** beim Deployment Ihrer Applikation sehen, so können Sie versuchen, diese mittels einer der folgenden Vorgehensweisen aufzulösen.



## WICHTIG

Applikationen, die Hibernate direkt mit Seam 2.2 verwenden, können eine innerhalb der Applikation gepackte Version von Hibernate 3 verwenden. Hibernate 4, welches mittels des org.hibernate Moduls der JBoss EAP 6 bereitgestellt wird, wird von Seam 2.2 nicht unterstützt. Dieses Beispiel soll Ihnen dabei helfen, Ihre Applikation auf der JBoss EAP 6 in Betrieb zu nehmen. Bitte beachten Sie, dass das Packen von Hibernate 3 mit einer Seam 2.2 Applikation keine unterstützte Konfiguration ist.

### Prozedur 3.13. Konfiguration der Applikation

#### 1. Kopieren Sie die benötigten Hibernate 3 JARs in Ihre Applikationsbibliothek.

Sie können dieses Problem möglicherweise beheben, indem Sie die betreffenden, die fehlenden Klassen enthaltenden Hibernate 3 JARs in das **lib/-**Verzeichnis kopieren oder indem Sie diese mittels einer anderen Methode dem Klassenpfad hinzufügen. In einigen Fällen kann dies aufgrund der Verwendung unterschiedlicher Hibernate-Versionen zu einer **ClassCastException** oder anderen Problemen beim Klassenladen führen. Ist dies der Fall, so verwenden Sie die nächste Vorgehensweise.

#### 2. Weisen Sie den Server dazu an, nur die Hibernate 3 Bibliotheken zu benutzen.

Die JBoss EAP 6 gestattet es Ihnen, Hibernate 3.5 (oder höher) Persistenz-Provider-Jars mit der Applikation zu verpacken. Um den Server dazu anzuweisen, nur die Hibernate 3 Bibliotheken zu verwenden und die Hibernate 4 Bibliotheken auszuschließen, müssen Sie das **jboss.as.jpa.providerModule** in der **persistence.xml** wie folgt auf **hibernate3-bundled** setzen:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
version="1.0">
  <persistence-unit name="plannerdatasource_pu">
    <description>Hibernate 3 Persistence Unit.</description>
    <jta-data-source>java:jboss/datasources/PlannerDS</jta-data-
source>
    <properties>
      <property name="hibernate.show_sql" value="false" />
      <property name="jboss.as.jpa.providerModule"
value="hibernate3-bundled" />
    </properties>
  </persistence-unit>
</persistence>
```

Der Java Persistence API (JPA) Deployer spürt das Vorhandensein des Persistenz-Providers in der Applikation auf und wird die Hibernate 3 Bibliotheken verwenden. Weitere Informationen zu JPA Persistenz-Properties finden Sie unter [Abschnitt 3.2.2.3, »Properties der Persistenzeinheit«](#).

#### 3. Deaktivieren des Caches der zweiten Ebene von Hibernate

Das Cache der zweiten Ebene bei Hibernate 3 zeigt mit der JBoss EAP 6 nicht dasselbe Verhalten wie frühere Releases. Falls Sie mit Ihrer Applikation das Hibernate Cache der zweiten Ebene Verwenden, so müssen Sie es bis zum Upgrade auf Hibernate 4 deaktivieren. Um das

Cache der zweiten Ebene zu deaktivieren setzen Sie `<hibernate.cache.use_second_level_cache>` in der `persistence.xml`-Datei auf `false`.

[Fehler melden](#)

### 3.2.2.3. Properties der Persistenzeinheit

#### Konfigurations-Properties von Hibernate 4.x

Die JBoss EAP 6 stellt automatisch die folgenden Hibernate 4.x Konfigurations-Properties ein:

**Tabelle 3.4. Properties der Hibernate Persistenzeinheit**

Property-Name	Standardwert	Zweck
<code>hibernate.id.new_generator_mappings</code>	true	<p>Diese Einstellung ist wichtig, wenn Sie <code>@GeneratedValue(AUTO)</code> zur Generierung eindeutiger Indexschlüsselwerte für neue Entities verwenden. Neue Applikationen sollten den Standardwert von <b>true</b> beibehalten. Bestehende Applikationen, die Hibernate 3.3.x verwenden, müssen es vielleicht auf <b>false</b> umstellen, um weiterhin ein Sequenzobjekt oder einen tabellenbasierten Generator zu verwenden und Rückwärtskompatibilität beizubehalten. Die Applikation kann diesen Wert in der <code>persistence.xml</code>-Datei außer Kraft setzen.</p> <p>Weitere Informationen zu diesem Verhalten finden Sie unten.</p>
<code>hibernate.transaction.jta.platform</code>	Instanz des <code>org.hibernate.service.jta.platform.spi.JtaPlatform</code> -Interface	Diese Klasse gibt den Transaction-Manager, die Benutzer-Transaction und die Transaction Synchronization Registry an Hibernate.
<code>hibernate.ejb.resource_scanner</code>	Instanz des <code>org.hibernate.ejb.packaging.Scanner</code> -Interface	Diese Klasse weiß, wie der JBoss EAP 6 Annotationsindexer für ein schnelleres Deployment eingesetzt wird.
<code>hibernate.transaction.manager_lookup_class</code>		Diese Property wird entfernt, falls sie in der <code>persistence.xml</code> aufgefunden wird, da sie mit <code>hibernate.transaction.jta.platform</code> in Konflikt stehen kann.
<code>hibernate.session_factory_name</code>	<code>QUALIFIED_PERSISTENCE_UNIT_NAME</code>	Dies wird auf den Applikationsnamen + Persistenzeinheitsnamen eingestellt. Die Applikation kann einen anderen Wert festlegen, aber dieser muss über alle Applikations-Deployments auf der JBoss EAP 6 Instanz hinweg eindeutig sein.



Property-Name	Standardwert	Zweck
<b>hibernate.session_factory_name_is_jndi</b>	false	Dies wird nur eingestellt, falls die Applikation keinen Wert für den <b>hibernate.session_factory_name</b> eingestellt hat.
<b>hibernate.ejb.entitymanager_factory_name</b>	<i>QUALIFIED_PERSISTENCE_UNIT_NAME</i>	Dies wird auf den Applikationsnamen + Persistenzeinheitennamen eingestellt. Die Applikation kann einen anderen Wert festlegen, aber dieser muss über alle Applikations-Deployments auf der JBoss EAP 6 Instanz hinweg eindeutig sein.

In Hibernate 4.x, wenn **new\_generator\_mappings** auf **true** eingestellt ist:

- **@GeneratedValue(AUTO)** mappt zu **org.hibernate.id.enhanced.SequenceStyleGenerator**.
- **@GeneratedValue(TABLE)** mappt zu **org.hibernate.id.enhanced.TableGenerator**.
- **@GeneratedValue(SEQUENCE)** mappt zu **org.hibernate.id.enhanced.SequenceStyleGenerator**.

In Hibernate 4.x, wenn **new\_generator\_mappings** auf **false** eingestellt ist:

- **@GeneratedValue(AUTO)** mappt zu Hibernate "native".
- **@GeneratedValue(TABLE)** mappt zu **org.hibernate.id.MultipleHiLoPerTableGenerator**.
- **@GeneratedValue(SEQUENCE)** mappt zu Hibernate "seqhilo".

Weitere Informationen zu diesen Properties finden Sie unter <http://www.hibernate.org/docs> im [Hibernate 4.1 Developer Guide](#).

### JPA Persistenz-Properties

Die folgenden JPA-Properties werden in der Persistenzeinheitdefinition in der **persistence.xml**-Datei unterstützt:

**Tabelle 3.5. Properties der JPA-Persistenzeinheit**

Property-Name	Standardwert	Zweck
<b>jboss.as.jpa.providerModule</b>	<b>org.hibernate</b>	<p>Der Name des Persisten-Provider-Moduls.</p> <p>Der Wert sollte <b>hibernate3-bundled</b> sein, falls Hibernate 3 JARs im Applikationsarchiv sind.</p> <p>Ist ein Persistenz-Provider in die Applikation gepackt, so sollte der Wert <b>application</b> lauten.</p>

Property-Name	Standardwert	Zweck
<b>jboss.as.jpa.adapterModule</b>	<b>org.jboss.as.jpa.hibernate:4</b>	<p>Der Name der Integrationsklassen, die der JBoss EAP 6 bei der Arbeit mit dem Persistenz-Provider helfen.</p> <p>Die derzeit gültigen Werte sind:</p> <ul style="list-style-type: none"> <li>• <b>org.jboss.as.jpa.hibernate:4</b>: Dies ist für die Hibernate 4 Integrationsklassen</li> <li>• <b>org.jboss.as.jpa.hibernate:3</b>: Dies ist für die Hibernate 3 Integrationsklassen</li> </ul>

[Fehler melden](#)

### 3.2.2.4. Aktualisieren Sie Ihre Hibernate 3 Applikation, damit sie Hibernate 4 benutzt

#### Zusammenfassung

Wenn Sie Ihre Applikation für die Verwendung von Hibernate 4 aktualisieren, so sind einige der Updates allgemein und gelten unabhängig von der zum aktuellen Zeitpunkt verwendeten Version von Hibernate. Bei anderen Updates müssen Sie bestimmen, welche Version die Applikation derzeit verwendet.

#### Prozedur 3.14. Aktualisieren Sie die Applikation, damit sie Hibernate 4 benutzt

1. Das Standardverhalten des selbstinkrementierenden Sequenzgenerators hat sich bei der JBoss EAP 6 geändert. Für weitere Informationen, siehe [Abschnitt 3.2.2.5, »Beibehaltung des bestehenden Verhaltens des selbstgenerierten Wertes der Hibernate-Identität«](#).
2. Bestimmen Sie die derzeit von der Applikation verwendete Version von Hibernate und wählen Sie unten den passenden Aktualisierungsvorgang aus.
  - [Abschnitt 3.2.2.6, »Migration Ihrer Hibernate 3.3.x Applikation zu Hibernate 4.x«](#)
  - [Abschnitt 3.2.2.7, »Migration Ihrer Hibernate 3.5.x Applikation zu Hibernate 4.x«](#)
3. Sie [Abschnitt 3.2.2.8, »Bearbeiten Sie Persistenz-Properties für migrierte Seam- und Hibernate-Applikationen, die in einer geclusterten Umgebung laufen«](#) falls Ihre Applikation in einer geclusterten Umgebung laufen wird.

[Fehler melden](#)

### 3.2.2.5. Beibehaltung des bestehenden Verhaltens des selbstgenerierten Wertes der Hibernate-Identität

Mit Hibernate 3.5 wurde eine neue Kern-Property namens **hibernate.id.new\_generator\_mappings** vorgestellt, die steuert, wie Identitäts- oder Sequenzspalten bei der Verwendung von **@GeneratedValue** generiert werden. Bei der JBoss EAP 6 wird der Standardwert für diese Property wie folgt eingestellt:

- Wenn Sie eine native Hibernate Applikation deployen, so lautet der Standardwert **false**.
- Wenn Sie eine JPA-Applikation deployen, so lautet der Standardwert **true**.

#### Richtlinien für neue Applikationen

Neue Applikationen, die die `@GeneratedValue`-Annotation verwenden, sollten den Wert der `hibernate.id.new_generator_mappings`-Property auf `true` einstellen. Dies ist die bevorzugte Einstellung, da Sie über verschiedene Datenbanken hinweg portierbarer ist. In den meisten Fällen ist dies effizienter und in manchen Fällen geht es die Kompatibilität mit der JPA 2 Spezifikation an.

- Für neue JPA-Applikationen besitzt JBoss EAP 6 für die `hibernate.id.new_generator_mappings`-Property die Standardeinstellung `true`, und dies sollte nicht geändert werden.
- Für neue native Hibernate Applikationen besitzt JBoss EAP 6 für die `hibernate.id.new_generator_mappings`-Property die Standardeinstellung `false`. Sie sollten diese Property auf `true` einstellen.

### Richtlinien für bestehende JBoss EAP 5 Applikationen

Bestehende Applikationen, die die `@GeneratedValue`-Annotation verwenden sollten sicherstellen, dass derselbe Generator zur Erstellung der Primärschlüsselwerte für neue Entities verwendet wird, wenn die Applikation zur JBoss EAP 6 migriert wird.

- Für bestehende JPA-Applikationen besitzt JBoss EAP 6 für die `hibernate.id.new_generator_mappings`-Property die Standardeinstellung `true`. Sie sollten diese Property in der `persistence.xml`-Datei auf `false` einstellen.
- Für bestehende native Hibernate Applikationen besitzt JBoss EAP 6 für `hibernate.id.new_generator_mappings` die Standardeinstellung `false`, und dies sollte nicht geändert werden.

Weitere Informationen zu diesen Property-Einstellungen finden Sie unter [Abschnitt 3.2.2.3, »Properties der Persistenzeinheit«](#).

[Fehler melden](#)

### 3.2.2.6. Migration Ihrer Hibernate 3.3.x Applikation zu Hibernate 4.x

#### 1. Mappen Sie Hibernate text-Typen zu JDBC LONGVARCHAR

In Hibernate-Versionen vor 3.5 war der `text`-Typ zu `JDBC CLOB` gemappt. Ein neuer Hibernate-Typ namens `materialized_clob` ist in Hibernate 4 dazugekommen und mappt Java `String`-Properties zu `JDBC CLOB`. Falls Ihre Applikation als `type="text"` konfigurierte Properties besitzt, die zu `JDBC CLOB` gemappt werden sollen, so müssen Sie einen der folgenden Schritte durchführen:

- Falls Ihre Applikation hbm-Mapping-Dateien verwendet, so ändern Sie die Property zu `type="materialized_clob"`.
- Falls Ihre Applikation Annotationen verwendet, so sollten Sie `@Type(type = "text")` durch `@Lob` ersetzen.

#### 2. Überprüfen Sie den Code, um Änderungen in den wiedergegebenen Wertetypen zu finden

Numerische aggregierte Criteria Projections geben jetzt denselben Wertetyp wie ihre HQL-Entsprechungen wieder. Als Folge haben sich die Wiedergabetypen folgender Projections in `org.hibernate.criterion` geändert.

- Aufgrund der Änderungen in `CountProjection`, `Projections.rowCount()`, `Projections.count(propertyName)` und `Projections.countDistinct(propertyName)`, geben die `count`- und `count`

**distinct**-Projections jetzt einen **Long**-Wert wieder.

- b. Aufgrund von Änderungen in **Projections.sum(propertyName)** geben die **sum**-Projections einen Wertetyp wieder, der vom Property-Typ abhängt.



#### ANMERKUNG

Falls Sie Ihren Applikationscode nicht bearbeiten, so kann dies zu einer **java.lang.ClassCastException** führen.

- i. Für als Long, Short, Integer oder primitive gemappte Integertypen wird ein Long-Wert wiedergegeben;
- ii. Für als Float, Double oder primitive Floating-Point-Typen gemappte Properties wird ein Double-Wert wiedergegeben.

[Fehler melden](#)

### 3.2.2.7. Migration Ihrer Hibernate 3.5.x Applikation zu Hibernate 4.x

1. Fügen Sie die AnnotationConfiguration in die Konfiguration ein.

Obwohl **AnnotationConfiguration** jetzt veraltet ist, sollte dies Ihre Migration nicht beeinflussen.

Falls Sie noch eine **hbm.xml**-Datei verwenden so sollten Sie sich dessen bewusst sein, dass die JBoss EAP 6 jetzt **org.hibernate.cfg.EJB3NamingStrategy** in **AnnotationConfiguration** statt der **org.hibernate.cfg.DefaultNamingStrategy** aus früheren Releases verwendet. Falls Ihre Naming-Strategy den Namen einer Assoziationstabelle bestimmt (many-to-many und Collections von Elementen), so begegnen Sie möglicherweise diesem Problem. Um es zu lösen können Sie Hibernate dazu anweisen, die Legacy **org.hibernate.cfg.DefaultNamingStrategy** durch Aufrufen von **Configuration#setNamingStrategy** und deren Weitergabe an **org.hibernate.cfg.DefaultNamingStrategy#INSTANCE** zu verwenden.

2. Bearbeiten Sie die Namespaces, damit sie mit den neuen Hibernate DTD Dateinamen konform sind, wie in der nachstehenden Tabelle ausgewiesen.

**Tabelle 3.6. DTD Namespace Mapping Tabelle**

Früherer DTD Namespace	Neuer DTD Namespace
<code>http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd</code>	<code>http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd</code>
<code>http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd</code>	<code>http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd</code>

3. Bearbeiten Sie die Umgebungsvariablen.
  - a. Falls Sie Oracle und die **materialized\_clob**- oder **materialized\_blob**-Properties verwenden, so muss die allgemeine Umgebungsvariable **hibernate.jdbc.use\_streams\_for\_binary** auf true eingestellt sein.

- b. Falls Sie PostgreSQL und die **CLOB** oder **BLOB**-Properties verwenden, so muss die allgemeine Umgebungsvariable `hibernate.jdbc.use_streams_for_binary` auf `false` eingestellt sein.

[Fehler melden](#)

### 3.2.2.8. Bearbeiten Sie Persistenz-Properties für migrierte Seam- und Hibernate-Applikationen, die in einer geclusterten Umgebung laufen

Falls Sie eine JPA Container-gemanagte Applikation migrieren, so sollten keine Serialisierungsprobleme auftreten, da die erweiterten Persistenzkontexte automatisch an den Container gegeben werden.

Aufgrund von Änderungen bei Hibernate kann es jedoch zu Serialisierungsproblemen kommen, wenn Sie Ihre migrierte Seam- oder Hibernate-Applikation in einer geclusterten Umgebung laufen lassen. Dabei können Fehlerprotokollmeldungen wie die folgende erscheinen:

```
javax.ejb.EJBTransactionRolledbackException: JBAS010361: Failed to
deserialize
....
Caused by: java.io.InvalidObjectException: could not resolve session
factory during session deserialization
[uuid=8aa29e74373ce3a301373ce3a44b0000, name=null]
```

Um diese Fehler zu beheben müssen Sie Properties in der Konfigurationsdatei bearbeiten. In den meisten Fällen ist dies die **persistence.xml**-Datei. Für native Hibernate API-Applikationen ist es die **hibernate.cfg.xml**-Datei.

#### Prozedur 3.15. Stellen Sie Persistenz-Properties zum Betrieb in einer geclusterten Umgebung ein

1. Setzen Sie den **hibernate.session\_factory\_name**-Wert auf einen eindeutigen Namen. Dieser Name muss über alle Applikations-Deployments auf der JBoss EAP 6 Instanz hinweg eindeutig sein. Zum Beispiel:

```
<property name="hibernate.session_factory_name" value="jboss-seam-
booking.ear_session_factory"/>
```

2. Setzen Sie den **hibernate.ejb.entitymanager\_factory\_name**-Wert auf einen eindeutigen Namen. Dieser Name muss über alle Applikations-Deployments auf der JBoss EAP 6 Instanz hinweg eindeutig sein. Zum Beispiel:

```
<property name="hibernate.ejb.entitymanager_factory_name"
value="seam-booking.ear_PersistenceUnitName"/>
```

Weitere Informationen zu den Einstellungen der Hibernate JPA Persistenzeinheit-Property finden Sie unter [Abschnitt 3.2.2.3, »Properties der Persistenzeinheit«](#).

[Fehler melden](#)

### 3.2.2.9. Aktualisieren Sie Ihre Applikation, damit sie mit der JPA 2.0 Spezifikation konform ist

#### Zusammenfassung

Die JPA 2.0 Spezifikation erfordert, dass ein Persistenzkontext nicht außerhalb einer JTA-Transaktion

fortgepflanzt werden kann. Falls Ihre Application nur transaktionsbegrenzte Persistenzkontexte verwendet, so ist das Verhalten bei der JBoss EAP 6 dasselbe wie in früheren Versionen des Applikationsservers, und es sind keine Änderungen notwendig. Falls Ihre Applikation jedoch einen erweiterten Persistenzkontext ("extended persistence context" oder kurz XPC) verwendet, um Warteschlangen oder das Batching von Datenänderungen zu ermöglichen, so kann es sein, dass Sie Änderungen an Ihrer Applikation vornehmen müssen.

### Fortpflanzungsverhalten des Persistenzkontext

Falls Ihre Applikation ein stateful Session Bean besitzt, **Bean1**, das einen erweiterten Persistenzkontext besitzt, und es ruft ein stateless Session Bean, **Bean2** auf, das einen transaktionsbegrenzten Persistenzkontext verwendet, so können Sie das folgende Verhalten erwarten:

- Falls **Bean1** eine JTA-Transaction startet und den **Bean2**-Methodenaufruf mit der JTA-Transaction aktiviert, so ist das Verhalten bei der JBoss EAP 6 dasselbe wie in früheren Releases und es ist keine Änderung notwendig.
- Falls **Bean1** keine JTA-Transaction startet und den **Bean2**-Methodenaufruf macht, so pflanzt die JBoss EAP 6 den erweiterten Persistenzkontext nicht in **Bean2** fort. Dieses Verhalten unterscheidet sich von dem anderer Releases, die den erweiterten Persistenzkontext in **Bean2** fortpflanzen. Falls Ihre Applikation erwartet, dass der erweiterte Persistenzkontext in das Bean mit dem transaktionalen Entity-Manager fortgepflanzt wird, so müssen Sie Ihre Applikation dahingehend ändern, dass der Aufruf innerhalb einer aktiven JTA-Transaction erfolgt.

[Fehler melden](#)

### 3.2.2.10. Ersetzen des Caches der zweiten Ebene von JPA/Hibernate durch Infinispan

#### Zusammenfassung

Das JBoss Cache wurde für das Cache der zweiten Ebene (2LC) durch Infinispan ersetzt. Dies erfordert eine Änderung an der **persistence.xml**-Datei. Die Syntax ist etwas anders, je nachdem ob Sie JPA oder das Hibernate Cache der zweiten Ebene verwenden. Diese Beispiele gehen davon aus, dass Sie Hibernate verwenden.

Dies ist ein Beispiel wie Properties für das Cache der zweiten Ebene in der **persistence.xml**-Datei bei der JBoss EAP 5.x spezifiziert wurden.

```
<property name="hibernate.cache.region.factory_class"
value="org.hibernate.cache.jbc2.JndiMultiplexedJBossCacheRegionFactory"/>
<property name="hibernate.cache.region.jbc2.cachefactory"
value="java:CacheManager"/>
<property name="hibernate.cache.use_second_level_cache" value="true"/>
<property name="hibernate.cache.region.jbc2.cfg.entity" value="mvcc-
entity"/>
<property name="hibernate.cache.region_prefix" value="services"/>
```

Die folgenden Schritte verwenden dieses Beispiel zur Konfiguration von Infinispan in der JBoss EAP 6.

#### Prozedur 3.16. Bearbeiten Sie die **persistence.xml**-Datei zur Verwendung von Infinispan

##### 1. Konfiguration von Infinispan für eine JPA-Applikation in der JBoss EAP 6

Auf diese Weise legen Sie Properties fest, um dieselbe Konfiguration für eine JPA-Applikation mittels Infinispan bei der JBoss EAP 6 zu erreichen:

```
<property name="hibernate.cache.use_second_level_cache"
value="true"/>
```

Zusätzlich müssen Sie einen **shared-cache-mode** mit einem Wert von **ENABLE\_SELECTIVE** oder **ALL** wie folgt festlegen:

- **ENABLE\_SELECTIVE** ist der Standard und der empfohlene Wert. Es bedeutet, dass Entities nicht gecacht werden, wenn Sie diese nicht explizit als cachbar kennzeichnen.

```
<shared-cache-mode>ENABLE_SELECTIVE</shared-cache-mode>
```

- **ALL** bedeutet, dass Entities immer gecacht werden, selbst wenn Sie diese als nicht cachbar kennzeichnen.

```
<shared-cache-mode>ALL</shared-cache-mode>
```

## 2. Konfiguration von Infinispan für eine native Hibernate Applikation in der JBoss EAP 6

Auf diese Weise legen Sie dieselbe Konfiguration für eine native Hibernate Applikation mittels Infinispan bei der JBoss EAP 6 fest:

```
<property name="hibernate.cache.region.factory_class"
value="org.jboss.as.jpa.hibernate4.infinispan.InfinispanRegionFactor
y"/>
<property name="hibernate.cache.infinispan.cachemanager"
value="java:jboss/infinispan/container/hibernate"/>
<property name="hibernate.transaction.manager_lookup_class"
value="org.hibernate.transaction.JBossTransactionManagerLookup"/>
<property name="hibernate.cache.use_second_level_cache"
value="true"/>
```

Sie müssen der **MANIFEST.MF**-Datei die folgenden Abhängigkeiten hinzufügen:

```
Manifest-Version: 1.0
Dependencies: org.infinispan, org.hibernate
```

Weitere Informationen zu Hibernate Cache-Properties finden Sie unter: [Abschnitt 3.2.2.11, »Hibernate Cache Properties«](#).

[Fehler melden](#)

### 3.2.2.11. Hibernate Cache Properties

Tabelle 3.7. Properties

Property-Name	Beschreibung
<b>hibernate.cache.region.factory_class</b>	Der Klassenname eines benutzerdefinierten <b>CacheProvider</b> .



Property-Name	Beschreibung
<code>hibernate.cache.use_minimal_puts</code>	Boolean. Optimiert die Operationsweise des Cache der zweiten Ebene um Schreibvorgänge zu minimieren zu Gunsten häufigerer Lesevorgänge. Diese Einstellung ist besonders bei geclusterten Caches nützlich und bei Hibernate3 für geclusterte Cache-Implementierungen standardmäßig aktiviert.
<code>hibernate.cache.use_query_cache</code>	Boolean. Aktiviert das Anfragen-Cache. Individuelle Anfragen müssen immer noch als cachbar eingestellt werden.
<code>hibernate.cache.use_second_level_cache</code>	Boolean. Verwendet zur kompletten Deaktivierung des Cache der zweiten Ebene, welches standardmäßig für Klassen aktiviert ist, die ein <b>&lt;cache&gt;</b> -Mapping festlegen.
<code>hibernate.cache.query_cache_factory</code>	Der Klassenname eines benutzerdefinierten <b>QueryCache</b> -Interface. Der Standardwert ist das eingebaute <b>StandardQueryCache</b> .
<code>hibernate.cache.region_prefix</code>	Ein Präfix zur Verwendung von Regionsnamen des Chache der zweiten Ebene.
<code>hibernate.cache.use_structured_entries</code>	Boolean. Zwingt Hibernate zur Speicherung von Daten in benutzerfreundlicherem Format im Cache der zweiten Ebene.
<code>hibernate.cache.default_cache_concurrency_strategy</code>	Einstellung wird für den Namen der standardmäßigen <b>org.hibernate.annotations.CacheConcurrencyStrategy</b> verwendet, die benutzt werden soll wenn entweder <b>@Cacheable</b> oder <b>@Cache</b> verwendet werden. <b>@Cache(strategy="...")</b> wird zur Außerkraftsetzung dieses Standards verwendet.

[Fehler melden](#)

### 3.2.2.12. Migration zu Hibernate Validator 4

#### Zusammenfassung

Bei Hibernate Validator 4.x handelt es sich um eine komplett neue Code-Basis, die die [JSR 303 - Bean-Validierung implementiert](#). Der Migrationsprozess vom Validator 3.x zu 4.x ist recht unkompliziert, aber es gibt einige Änderungen, die Sie bei der Migration Ihrer Applikation beachten müssen.

#### Prozedur 3.17. Sie müssen eine oder mehrere der folgenden Aufgaben ausführen

1. **Zugreifen auf die standardmäßige ValidatorFactory**

Die JBoss EAP 6 bindet eine standardmäßige ValidatorFactory unter dem Namen `java:comp/ValidatorFactory` an den JNDI-Kontext.

2. **Vom Lebenszyklus ausgelöste Validierung verstehen**



Wenn in Verbindung mit Hibernate Core 4 eingesetzt, so wird die Lebenszyklus basierte Validierung automatisch durch Hibernate Core aktiviert.

- a. Die Validierung erfolgt an Entity **INSERT**, **UPDATE** und **DELETE**-Operationen.
- b. Sie können die nach Ereignistyp zu validierenden Gruppen mittels der folgenden Properties konfigurieren:

- **javax.persistence.validation.group.pre-persist**,
- **javax.persistence.validation.group.pre-update**, and
- **javax.persistence.validation.group.pre-remove**.

Die Werte dieser Properties sind durch Kommas getrennt, voll qualifizierte Klassennamen der zu validierenden Gruppen.

Validierungsgruppen sind ein neues Feature der Bean Validierungsspezifikation. Falls Sie dieses neue Feature nicht nutzen möchten, so sind bei der Migration zum Hibernate Validator 4 keine Änderungen notwendig.

- c. Sie können die Lebenszyklus-basierte Validierung durch Einstellung der **javax.persistence.validation.mode**-Property auf **none** deaktivieren. Andere gültige Werte für diese Property sind **auto** (der Standard), **callback** und **ddl**.

### 3. Konfigurieren Sie Ihre Applikation zur Verwendung manueller Validierung

- a. Falls Sie die Validierung manuell steuern möchten, so können Sie auf eine der folgenden Weisen einen Validator erstellen:
  - Erstellen Sie eine **Validator**-Instanz aus der **ValidatorFactory** mittels der **getValidator()**-Methode.
  - Speisen Sie die Validator-Instanzen in Ihr EJB, Ihr CDI-Bean oder eine andere einspeisbare Java EE Ressource ein.
- b. Sie können den von der **ValidatorFactory.usingContext()** wiedergegebenen **ValidatorContext** zur Anpassung Ihrer Validator-Instanz verwenden. Mittels dieses API können Sie benutzerdefinierte **MessageInterpolator**, **TraversableResolver** und **ConstraintValidatorFactory** konfigurieren. Diese Interfaces sind in der Bean Validierungsspezifikation festgelegt und neu im Hibernate Validator 4.

### 4. Bearbeiten Sie den Code, der mit den neuen Bean Validierungsbedingungen verwendet werden soll

Die neuen Validierungsbedingungen auf Bean-Ebene erfordern Code-Änderungen wenn Sie zum Hibernate Validator 4 migrieren.

- a. Um ein Upgracde zu Hibernate Validator 4 durchzuführen müssen Sie die Bedingungen in den folgenden Paketen Verwenden:
  - **javax.validation.constraints**
  - **org.hibernate.validator.constraints**
- b. Alle Bedingungen, die im Hibernate Validator 3 vorhanden waren, gibt es auch im Hibernate Validator 4. Um diese zu verwenden, müssen Sie die festgelegte Klasse importieren und - in manchen Fällen - den Namen oder den Typ des Bedingungsparameters ändern.

## 5. Verwendung benutzerdefinierter Bedingungen

Beim Hibernate Validator 3 musste eine benutzerdefinierte Bedingung im **org.hibernate.validator.Validator**-Interface implementiert werden. Beim Hibernate Validator 4 müssen Sie das **javax.validation.ConstraintValidator**-Interface implementieren. Dieses Interface enthält dieselben **initialize()**- und **isValid()**-Methoden wie das vorherige Interface, wobei sich allerdings die Methodensignatur geändert hat. Desweiteren wird **DDL**-Änderung beim Hibernate Validator 4 nicht mehr unterstützt.

[Fehler melden](#)

## 3.2.3. JSF-Änderungen

### 3.2.3.1. Applikationen die Verwendung älterer Versionen von JSF ermöglichen

#### Zusammenfassung

Falls Ihre Applikation eine ältere Version von JSF verwendet, so müssen Sie kein Upgrade zu JSF 2.0 durchführen. Sie können stattdessen eine **jboss-deployment-structure.xml**-Datei erstellen, damit die JBoss EAP 6 JSF 1.2 statt JSF 2.0 mit dem Deployment Ihrer Applikation verwendet. Dieser JBoss spezifische Deployment-Deskriptor wird zur Steuerung des Klassenladens verwendet und wird im **META-INF/-** oder **WEB-INF/-** Verzeichnis Ihres WAR oder im **META-INF/-** Verzeichnis Ihres EAR platziert.

Nachfolgend sehen Sie das Beispiel einer **jboss-deployment-structure.xml**-Datei, die eine Abhängigkeit für das JSF 1.2 Modul hinzufügt und das automatische Laden für das JSF 2.0 verhindert.

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.0">
  <deployment>
    <dependencies>
      <module name="javax.faces.api" slot="1.2" export="true"/>
      <module name="com.sun.jsf-impl" slot="1.2" export="true"/>
    </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
      <module name="javax.faces.api" slot="main"/>
      <module name="com.sun.jsf-impl" slot="main"/>
    </exclusions>
    <dependencies>
      <module name="javax.faces.api" slot="1.2"/>
      <module name="com.sun.jsf-impl" slot="1.2"/>
    </dependencies>
  </sub-deployment>
</jboss-deployment-structure>
```

[Fehler melden](#)

## 3.2.4. Web-Dienste Änderungen

### 3.2.4.1. Web-Dienste Änderungen

JBoss EAP 6 beinhaltet Support für das Deployment der JAX-WS Webdienst Endpunkte. Dieser Support wird durch JBossWS bereitgestellt. Weitere Informationen zu Webdiensten finden Sie im Kapitel mit dem Titel *JAX-WS Webdienste* im *Entwicklungshandbuch* für die JBoss EAP 6 unter

[https://access.redhat.com/site/documentation/JBoss\\_Enterprise\\_Application\\_Platform/](https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/).

JBossWS 4 beinhaltet folgende Änderungen, die Ihre Migration beeinflussen können.

### JBossWS API Änderungen

SPI und Common Komponenten wurden in JBossWS 4 refakturiert. Die folgende Tabelle listet API und Verpackungsänderungen, die die Migration Ihrer Applikation beeinflussen können.

**Tabelle 3.8. JBossWS API Änderungen**

Altes JAR	Altes Paket	Neues JAR	Neues Paket
JBossWS SPI	org.jboss.wsf.spi.annotation.*	JBossWS API	org.jboss.ws.api.annotation.*
JBossWS SPI	org.jboss.wsf.spi.binding.*	JBossWS API	org.jboss.ws.api.binding.*
JBossWS SPI	org.jboss.wsf.spi.management.recording.*	JBossWS API	org.jboss.ws.api.monitoring.*
JBossWS SPI	org.jboss.wsf.spi.tools.*	JBossWS API	org.jboss.ws.api.tools.*
JBossWS SPI	org.jboss.wsf.spi.tools.ant.*	JBossWS API	org.jboss.ws.tools.ant.*
JBossWS SPI	org.jboss.wsf.spi.tools.cmd.*	JBossWS API	org.jboss.ws.tools.cmd.*
JBossWS SPI	org.jboss.wsf.spi.util.ServiceLoader	JBossWS API	org.jboss.ws.api.util.ServiceLoader
JBossWS Common	org.jboss.wsf.common.*	JBossWS API	org.jboss.ws.common.*
JBossWS Common	org.jboss.wsf.common.handler.*	JBossWS API	org.jboss.ws.api.handler.*
JBossWS Common	org.jboss.wsf.common.addressing.*	JBossWS API	org.jboss.ws.api.addressing.*
JBossWS Common	org.jboss.wsf.common.DOMUtils	JBossWS API	org.jboss.ws.api.util.DOMUtils
JBossWS Native	org.jboss.ws.annotation.EndpointConfig	JBossWS API	org.jboss.ws.api.annotation.EndpointConfig
JBossWS Framework	org.jboss.wsf.framework.invocation.RecordingServerHandler	JBossWS Common	org.jboss.ws.common.invocation.RecordingServerHandler

### @WebContext Annotation

In JBossWS 3.4.x war diese Annotation als **org.jboss.wsf.spi.annotation.WebContext** in JBossWS SPI JAR verpackt. In JBossWS 4.0 wurde diese Annotation zum

**org.jboss.ws.api.annotation.WebContext** im JBossWS API JAR verschoben. Wenn Ihre Applikation veraltete Abhängigkeiten beinhaltet, müssen Sie die Importe und Abhängigkeiten in Ihrem Applikations-Sourcecode ersetzen und das neue JBossWS API JAR kompilieren.

Es gibt auch eine Änderung an einem Attribut, das nicht rückwärts kompatibel ist. Das **String[] virtualHosts**-Attribut wurde zu **String virtualHost** geändert. In JBoss EAP 6 können Sie nur einen virtuellen Host pro Deployment festlegen. Falls mehrere Webdienste die **@WebContext**-Annotation verwenden, so muss der virtualHost-Wert für alle im Deployment-Archiv definierten Endpunkte identisch sein.

## Endpunkt-Konfiguration

JBossWS 4.0 bietet Integration des JBoss Web Services Stacks mit den meisten Apache CXF Modulen. Die Integrationsebene ermöglicht die Nutzung von standardmäßigen Webdienst APIs, einschließlich JAX-WS. Sie ermöglicht Ihnen außerdem ohne komplexe Konfiguration oder Setup die Nutzung von Apache CXF erweiterten Features zusätzlich zu JBoss EAP 6 Container.

Das **webservice**-Untersystem in der Domain-Konfiguration der JBoss EAP 6 beinhaltet vordefinierte Endpunkt-Konfigurationen. Sie können auch Ihre eigenen, zusätzlichen Endpunkt-Konfigurationen definieren. Die **@org.jboss.ws.api.annotation.EndpointConfig**-Annotation wird zur Referenzierung einer gegebenen Endpoint-Konfiguration verwendet.

Weitere Informationen zur Konfiguration von Webdienst Endpunkten im JBoss Server finden Sie im Kapitel *JAX-WS Webdienste* im Entwicklungshandbuch für die JBoss EAP 6 unter [https://access.redhat.com/site/documentation/JBoss\\_Enterprise\\_Application\\_Platform/](https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/).

## jboss-webservices.xml Deployment-Deskriptor

JBossWS 4.0 führt einen neuen Deployment Deskriptor zur Konfiguration von Webdiensten ein. Die **jboss-webservices.xml**-Datei liefert zusätzliche Informationen zu dem gegebenen Deployment und ersetzt zum Teil die veraltete **jboss.xml**-Datei.

Für EJB Webdienst-Deployments ist der erwartete Speicherort für die **jboss-webservices.xml**-Deskriptordatei im **META-INF**/-Verzeichnis. Für in der WAR-Datei gebündelte POJO- und EJB-Webdienst Endpunkte ist der erwartete Speicherort der **jboss-webservices.xml**-Datei im **WEB-INF**/-Verzeichnis.

Nachfolgend sehen Sie ein Beispiel für eine **jboss-webservices.xml**-Deskriptordatei und eine Tabelle, die die Elemente beschreibt.

```
<webservices>
  <context-root>foo</context-root>
  <config-name>Standard WSSecurity Endpoint</config-name>
  <config-file>META-INF/custom.xml</config-file>
  <property>
    <name>prop.name</name>
    <value>prop.value</value>
  </property>
  <port-component>
    <ejb-name>TestService</ejb-name>
    <port-component-name>TestServicePort</port-component-name>
    <port-component-uri>/*</port-component-uri>
    <auth-method>BASIC</auth-method>
    <transport-guarantee>NONE</transport-guarantee>
    <secure-wsdl-access>true</secure-wsdl-access>
  </port-component>
```

```

    <webservice-description>
      <webservice-description-name>TestService</webservice-
description-name>
      <wsdl-publish-location>file:///bar/foo.wsdl</wsdl-publish-
location>
    </webservice-description>
  </webservices>

```

Tabelle 3.9. jboss-webservice.xml Datei Element Beschreibung

Element Name	Beschreibung
context-root	Zur Verwendnung zur Anpassung des Kontext Root des Webdienst Deployments.
config-name config-file	Wird zur Assoziierung eines Endpunkt-Deployments mit einer gegebenen Endpoint-Konfiguration verwendet. Endpunkt-Konfigurationen werden in der referenzierten Konfigurationsdatei oder im <b>webservices</b> -Untersystem der Domain Konfiguration festgelegt.
property	Wird zur Einstellung eines einfachen Property Name Wertepaars zur Konfiguration des Webdienst Stack Verhaltens verwendet.
port-component	Wird zur Anpassung der EJB Endpunkt Ziel URI oder Kunfiguration sicherheitsbezogener Properties verwendet.
webservice-description	Wird zur Anpassung oder Außerkraftsetzung des Webdienst WSDL publizierten Speicherorts verwendet.

[Fehler melden](#)

### 3.2.5. JAX-RS- und RESTEasy-Änderungen

#### 3.2.5.1. Konfiguration der JAX-RS- und RESTEasy-Änderungen

Die JBoss EAP 6 stellt RESTEasy automatisch ein, so dass Sie es nicht selbst konfigurieren müssen. Sie sollten daher jede bestehende RESTEasy-Konfiguration aus Ihrer **web.xml**-Datei entfernen und diese durch eine der folgenden drei Optionen ersetzen:

1. Machen Sie **javax.ws.rs.core.Application** zur Unterklasse und verwenden Sie die **@ApplicationPath**-Annotation.

Dies ist die einfachste Option und erfordert keine xml-Konfiguration. Machen Sie **javax.ws.rs.core.Application** einfach zu einer Unterklasse in Ihrer Applikation und und annotieren Sie sie mit dem Pfad, wo Ihre JAX-RS-Klassen verfügbar werden sollen. Zum Beispiel:

```

@ApplicationPath("/mypath")
public class MyApplication extends Application {
}

```

Im Beispiel oben sind Ihre JAX-RS-Ressourcen im Pfad **/MY\_WEB\_APP\_CONTEXT/mypath/** verfügbar.



#### ANMERKUNG

Beachten Sie, dass der Pfad als **/mypath**, und nicht als **/mypath/\*** festgelegt werden sollte. Es sollten kein Schrägstrich oder Sternchen folgen.

2. Machen Sie **javax.ws.rs.core.Application** zur Unterklasse und verwenden Sie die **web.xml**-Datei zur Einstellung des JAX-RS-Mappings.

Falls Sie die **@ApplicationPath**-Annotation nicht verwenden möchten, so müssen Sie die **javax.ws.rs.core.Application** dennoch zur Unterklasse machen. Anschließend stellen Sie das JAX-RS-Mapping in der **web.xml**-Datei ein. Zum Beispiel:

```
public class MyApplication extends Application {
}
```

```
<servlet-mapping>
  <servlet-name>com.acme.MyApplication</servlet-name>
  <url-pattern>/hello/*</url-pattern>
</servlet-mapping>
```

Im Beispiel oben sind die JAX-RS-Ressourcen im Pfad **/MY\_WEB\_APP\_CONTEXT/hello** verfügbar.



#### ANMERKUNG

Sie können diese Vorgehensweise zur Außerkraftsetzung eines mittels der **@ApplicationPath**-Annotation eingestellten Applikationspfads verwenden.

3. Bearbeiten Sie die **web.xml**-Datei.

Falls Sie **Application** nicht zur Unterklasse machen wollen, so können Sie das JAX-RS-Mapping wie folgt in der **web.xml**-Datei einstellen:

```
<servlet-mapping>
  <servlet-name>javax.ws.rs.core.Application</servlet-name>
  <url-pattern>/hello/*</url-pattern>
</servlet-mapping>
```

Im Beispiel oben sind die JAX-RS-Ressourcen im Pfad **/MY\_WEB\_APP\_CONTEXT/hello** verfügbar.



#### ANMERKUNG

Wenn Sie diese Option wählen, so müssen Sie nur das Mapping hinzufügen. Das entsprechende Servlet müssen Sie nicht hinzufügen. Der Server ist für die automatische Hinzufügung des Servlets verantwortlich.

## 3.2.6. Änderungen des LDAP-Sicherheitsbereichs

### 3.2.6.1. Konfiguration der Änderungen des LDAP-Sicherheitsbereichs

Bei der JBoss EAP 5 wurde der LDAP-Sicherheitsbereich in einem **<application-policy>**-Element in der **login-config.xml**-Datei konfiguriert. Bei der JBoss EAP 6 wird der LDAP-Sicherheitsbereich im **<security-domain>**-Element in der Serverkonfigurationsdatei konfiguriert. Bei einem Standalone Server ist dies die **standalone/configuration/standalone.xml**-Datei. Läuft Ihr Server in einer Managed Domain, so ist dies die **domain/configuration/domain.xml**-Datei.

Nachfolgend sehen Sie ein Beispiel für LDAP Sicherheitsbereich Konfiguration in der **login-config.xml**-Datei in JBoss EAP 5:

```
<application-policy name="mcp_ldap_domain">
  <authentication>
    <login-module code="org.jboss.security.auth.spi.LdapExtLoginModule"
flag="required">
      <module-option
name="java.naming.factory.initial">com.sun.jndi.ldap.LdapCtxFactory</modul
e-option>
      <module-option
name="java.naming.security.authentication">simple</module-option>
      ....
    </login-module>
  </authentication>
</application-policy>
```

Nachfolgend sehen Sie ein Beispiel für eine LDAP-Konfiguration in der Serverkonfigurationsdatei in der JBoss EAP 6:

```
<subsystem xmlns="urn:jboss:domain:security:1.2">
  <security-domains>
    <security-domain name="mcp_ldap_domain" cache-type="default">
      <authentication>
        <login-module code="org.jboss.security.auth.spi.LdapLoginModule"
flag="required">
          <module-option name="java.naming.factory.initial"
value="com.sun.jndi.ldap.LdapCtxFactory"/>
          <module-option name="java.naming.security.authentication"
value="simple"/>
          ...
        </login-module>
      </authentication>
    </security-domain>
  </security-domains>
</subsystem>
```



## ANMERKUNG

Der XML-Parser hat sich bei der JBoss EAP 6 geändert. Bei der JBoss EAP 5 wurden die Modulooptionen als Elementinhalt wie folgt festgelegt:

```
<module-option
  name="java.naming.factory.initial">com.sun.jndi.ldap.LdapCtxFactory</module-option>
```

Jetzt müssen die Modulooptionen wie folgt als Elementattribute mit "value=" as festgelegt werden:

```
<module-option name="java.naming.factory.initial"
  value="com.sun.jndi.ldap.LdapCtxFactory"/>
```

[Fehler melden](#)

## 3.2.7. HornetQ-Änderungen

### 3.2.7.1. Über HornetQ und NFS

In den meisten Fällen ist NFS keine geeignete Methode zur Speicherung von JMS Daten zur Verwendung mit HornetQ, wenn NIO als Journaltyp verwendet wird, aufgrund der Weise, wie der synchron Sperrmechanismus funktioniert. NFS kann jedoch unter bestimmten Umständen verwendet werden, ausschließlich auf Red Hat Enterprise Linux Servern. Dies ist wegen der von Red Hat Enterprise Linux verwendeten NFS-Implementierung der Fall.

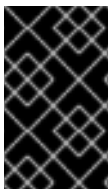
Die Red Hat Enterprise Linux NFS-Implementierung unterstützt sowohl direktes I/O (öffnet Dateien mit eingestelltem O\_DIRECT Flag) als auch Kernel-basiertes, asynchrones I/O. Mit beiden dieser Features vorhanden ist es möglich, NFS als eine geteilte Speicheroption zu verwenden, wobei strenge Konfigurationsregeln gelten:

- Das Red Hat Enterprise Linux NFS Client Cache muss deaktiviert sein.



## WICHTIG

Das Serverprotokoll sollte nach dem Start der JBoss EAP 6 geprüft werden um sicherzustellen, dass die native Bibliothek erfolgreich geladen wurde und dass der ASYNCIO Journaltyp verwendet wird. Lädt die native Bibliothek nicht, so schlägt HornetQ am NIO Journaltyp fehl und dies wird im Serverprotokoll angezeigt.



## WICHTIG

Die native Bibliothek, die asynchrones I/O implementiert macht es erforderlich, dass **libaio**, auf dem Red Hat Enterprise Linux System auf dem die JBoss EAP 6 läuft, installiert ist.

[Fehler melden](#)

### 3.2.7.2. Konfiguration einer JMS-Bridge zur Migration bestehender JMS-Nachrichten zur JBoss EAP 6



JBoss EAP 6 ersetzt JBoss Messaging durch HornetQ als standardmäßige JMS Implementierung. Die einfachste Weise der Migration von JMS Nachrichten aus der aktuellen Umgebung zu einer anderen ist die Verwendung einer JMS Bridge. Die Funktion einer JMS Bridge besteht im Konsumieren von Nachrichten von einer Quell-JMS Destination und deren Versand an eine Ziel-JMS Destination. Sie können eine JMS Bridge an einem JBoss EAP 5.x Server oder an einem JBoss EAP 6.1 oder späteren Server konfigurieren und deployen.

Details zur Migration von JMS Meldungen aus JBoss EAP 5.x to JBoss EAP 6.x finden Sie unter:

[Abschnitt 3.2.7.3, »JMS Bridge erstellen«](#)

[Fehler melden](#)

### 3.2.7.3. JMS Bridge erstellen

#### Zusammenfassung

Eine JMS Bridge liest Meldungen von einer Quellen JMS Warteschlange oder einem Thema und schickt sie an eine Ziel JMS Warteschlange oder ein Thema, die üblicherweise auf einem anderen Server sind. Dies kann benutzt werden um Meldungen zwischen jedem JMS Server zu überbrücken, solange sie JMS 1.1 kompatibel sind. Quellen und Ziel JMS Ressourcen können über JNDI gesucht werden und die Clientklassen für das JNDI Lookup müssen in einem Modul zusammengefasst sein. Der Modulname wird dann in der JMS Bridgekonfiguration deklariert.

#### Prozedur 3.18. JMS Bridge erstellen

Dieses Verfahren demonstriert, wie man eine JMS Bridge konfiguriert um Meldungen von einem JBoss EAP 5.x Server zu einem JBoss EAP 6 Server zu migrieren.

##### 1. Konfigurieren der Bridge auf dem Source JBoss EAP 5.x Server

Um Konflikte in Klassen zwischen Releases zu vermeiden, müssen Sie die folgenden Schritte zur Konfiguration der JMS Bridge auf JBoss EAP 5.x verwenden. Die Namen des SAR Verzeichnisses und der Bridge sind arbiträr und können auf Wunsch geändert werden.

- a. Erstellen Sie ein Unterverzeichnis im JBoss EAP 5 Deployment-Verzeichnis, das das SAR enthält, zum Beispiel: **EAP5\_HOME/server/PROFILE\_NAME/deploy/myBridge.sar/**.
- b. Erstellen Sie ein Unterverzeichnis namens **META-INF** in **EAP5\_HOME/server/PROFILE\_NAME/deploy/myBridge.sar/**.
- c. Erstellen Sie eine **jboss-service.xml** Datei im **EAP5\_HOME/server/PROFILE\_NAME/deploy/myBridge.sar/META-INF/** Verzeichnis. Sie sollte Informationen ähnlich dem folgendem Beispiel enthalten.

```
<server>
  <loader-repository>
    com.example:archive=unique-archive-name
    <loader-repository-
config>java2ParentDelegation=false</loader-repository-config>
  </loader-repository>

  <!-- JBoss EAP 6 JMS Provider -->
  <mbean code="org.jboss.jms.jndi.JMSProviderLoader"
name="jboss.messaging:service=JMSProviderLoader,name=EnterpriseAp
plicationPlatform6JMSProvider">
    <attribute
name="ProviderName">EnterpriseApplicationPlatform6JMSProvider</at
```

```

tribute>
    <attribute
name="ProviderAdapterClass">org.jboss.jms.jndi.JNDIProviderAdapte
r</attribute>
    <attribute
name="FactoryRef">jms/RemoteConnectionFactory</attribute>
    <attribute
name="QueueFactoryRef">jms/RemoteConnectionFactory</attribute>
    <attribute
name="TopicFactoryRef">jms/RemoteConnectionFactory</attribute>
    <attribute name="Properties">

java.naming.factory.initial=org.jboss.naming.remote.client.Initia
lContextFactory

java.naming.provider.url=remote://EnterpriseApplicationPlatform6h
ost:4447
    java.naming.security.principal=jbossuser
    java.naming.security.credentials=jbosspass
    </attribute>
</mbean>

    <mbean code="org.jboss.jms.server.bridge.BridgeService"
name="jboss.jms:service=Bridge,name=MyBridgeName" xmbean-
dd="xmdesc/Bridge-xmbean.xml">
    <depends optional-attribute-
name="SourceProviderLoader">jboss.messaging:service=JMSProviderLo
ader,name=JMSProvider</depends>
    <depends optional-attribute-
name="TargetProviderLoader">jboss.messaging:service=JMSProviderLo
ader,name=EnterpriseApplicationPlatform6JMSProvider</depends>
    <attribute
name="SourceDestinationLookup">/queue/A</attribute>
    <attribute
name="TargetDestinationLookup">jms/queue/test</attribute>
    <attribute name="QualityOfServiceMode">1</attribute>
    <attribute name="MaxBatchSize">1</attribute>
    <attribute name="MaxBatchTime">-1</attribute>
    <attribute name="FailureRetryInterval">60000</attribute>
    <attribute name="MaxRetries">-1</attribute>
    <attribute name="AddMessageIDInHeader">>false</attribute>
    <attribute name="TargetUsername">jbossuser</attribute>
    <attribute name="TargetPassword">jbosspass</attribute>
    </mbean>
</server>

```



## ANMERKUNG

Das **load-repository** ist präsent um sicherzustellen, dass das SAR einen isolierten Klassenlader besitzt. Beachten Sie auch, dass sowohl das JNDI Look-up als auch das Bridge "Ziel" Sicherheitsberechtigungen für den Benutzer "jbossuser" mit dem Passwort "jbosspass" beinhalten. Der Grund hierfür ist, dass JBoss EAP 6 standardmäßig gesichert ist. Der Benutzer namens "jbossuser" mit dem Passwort "jbosspass" wurde im **ApplicationRealm** mit der Rolle **guest** mittels **EAP\_HOME/bin/add\_user.sh**-Skript erstellt.

- d. Kopieren Sie die folgende JARs vom **EAP\_HOME/modules/system/layers/base/-** Verzeichnis in das **EAP5\_HOME/server/PROFILE\_NAME/deploy/myBridge.sar/-** Verzeichnis. Ersetzen Sie jede **VERSION\_NUMBER** durch die tatsächliche Versionsnummer in Ihrer JBoss EAP 6 Distribution.

- **org/hornetq/main/hornetq-core-VERSION\_NUMBER.jar**
- **org/hornetq/main/hornetq-jms-VERSION\_NUMBER.jar**
- **org/jboss/ejb-client/main/jboss-ejb-client-VERSION\_NUMBER.jar**
- **org/jboss/logging/main/jboss-logging-VERSION\_NUMBER.jar**
- **org/jboss/logmanager/main/jboss-logmanager-VERSION\_NUMBER.jar**
- **org/jboss/marshalling/main/jboss-marshalling-VERSION\_NUMBER.jar**
- **org/jboss/marshalling/river/main/jboss-marshalling-river-VERSION\_NUMBER.jar**
- **org/jboss/remote-naming/main/jboss-remote-naming-VERSION\_NUMBER.jar**
- **org/jboss/remoting3/main/jboss-remoting-VERSION\_NUMBER.jar**
- **org/jboss/sasl/main/jboss-sasl-VERSION\_NUMBER.jar**
- **org/jboss/netty/main/netty-VERSION\_NUMBER.jar**
- **org/jboss/remoting3/remote-jmx/main/remoting-jmx-VERSION\_NUMBER.jar**
- **org/jboss/xnio/main/xnio-api-VERSION\_NUMBER.jar**
- **org/jboss/xnio/nio/main.xnio-nio-VERSION\_NUMBER.jar**



## ANMERKUNG

Kopieren Sie das **EAP\_HOME/bin/client/jboss-client.jar** nicht einfach, da die javax API Klassen mit denen in JBoss EAP 5.x in Konflikt stehen.

## 2. Konfigurieren der Bridge auf dem Destination JBoss EAP 6 Server

Bei der JBoss EAP 6.1 und später kann die JMS Bridge zur Überbrückung von Nachrichten von jedem JMS 1.1 konformen Server verwendet werden. Weil die Quell- und Ziel JMS Ressourcen mittels JNDI aufgesucht werden, müssen die JNDI Lookup-Klassen des Quell-Messaging Providers oder Nachrichten-Brokers in einem JBoss Modul gebündelt sein. Die folgenden Schritte verwenden den fiktiven 'MyCustomMQ' Message Broker als ein Beispiel.

- a. Erstellen eines JBoss Moduls für den Messaging Provider.
  - i. Erstellen Sie eine Verzeichnisstruktur unter **EAP\_HOME/modules/system/layers/base/** für das neue Modul. Das **main/**-Unterverzeichnis enthält die Client JARs und die **module.xml**-Datei. Nachfolgend sehen Sie ein Beispiel der Verzeichnisstruktur, die für den MyCustomMQ Messaging Provider erstellt wurde:  
**EAP\_HOME/modules/system/layers/base/org/mycustommq/main/**
  - ii. Erstellen Sie im **main/**-Unterverzeichnis eine **module.xml**-Datei, die die Moduldefinition für den Messaging Provider enthält. Nachfolgend sehen Sie ein Beispiel der **module.xml**, die für den MyCustomMQ Messaging Provider erstellt wurde.

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.1" name="org.mycustommq">
  <properties>
    <property name="jboss.api" value="private"/>
  </properties>

  <resources>
    <!-- Insert resources required to connect to the
source or target -->
    <resource-root path="mycustommq-1.2.3.jar" />
    <resource-root path="mylogapi-0.0.1.jar" />
  </resources>

  <dependencies>
    <!-- Add the dependencies required by JMS Bridge code
-->
    <module name="javax.api" />
    <module name="javax.jms.api" />
    <module name="javax.transaction.api"/>
    <!-- Add a dependency on the org.hornetq module since
we send -->
    <!-- messages to the HornetQ server embedded in the
local EAP instance -->
    <module name="org.hornetq" />
  </dependencies>
</module>
```

- iii. Kopieren Sie die Messaging Provider JARs, die für den JNDI Lookup der Quellressourcen benötigt werden, in das **main/**-Unterverzeichnis des Moduls. Die Verzeichnisstruktur des MyCustomMQ Moduls sollte jetzt wie folgt aussehen.

```
modules/
  -- system
    -- layers
      -- base
        -- org
          -- mycustommq
```

```

`-- main
   |-- mycustommq-1.2.3.jar
   |-- mylogapi-0.0.1.jar
   |-- module.xml

```

b. Konfigurieren Sie die JMS Bridge im **messaging** Untersystem des JBoss EAP 6 Servers.

- i. Ehe Sie anfangen, stoppen Sie den Server und erstellen Sie eine Sicherungskopie der aktuellen Serverkonfigurationsdateien. Falls Sie einen Standalone Server betreiben ist dies die **EAP\_HOME/standalone/configuration/standalone-full-ha.xml**-Datei. In einer Managed Domain erstellen Sie eine Sicherungskopie von sowohl der **EAP\_HOME/domain/configuration/domain.xml**- als auch der **EAP\_HOME/domain/configuration/host.xml**-Datei.
- ii. Fügen Sie das **jms-bridge**-Element dem **messaging**-Subsystem in der Server Konfigurationsdatei hinzu. Die **source** und **<target>**-Elemente liefern die Namen der JMS-Ressourcen, die für JNDI Lookups verwendet werden. Falls **user** und **password** Berechtigungen festgelegt sind, so sind sie Argumente bei der Erstellung der JMS-Verbindung.

Nachfolgend sehen Sie ein Beispiel für das **jms-bridge**-Element, das für den MyCustomMQ Messaging Provider konfiguriert ist:

```

<subsystem xmlns="urn:jboss:domain:messaging:1.3">
    ...
    <jms-bridge name="myBridge" module="org.mycustommq">
        <source>
            <connection-factory name="ConnectionFactory"/>
            <destination name="sourceQ"/>
            <user>user1</user>
            <password>pwd1</password>
            <context>
                <property key="java.naming.factory.initial"
value="org.mycustommq.jndi.MyCustomMQInitialContextFactory"/>
                <property key="java.naming.provider.url"
value="tcp://127.0.0.1:9292"/>
            </context>
        </source>
        <target>
            <connection-factory name="java:/ConnectionFactory"/>
            <destination name="/jms/targetQ"/>
        </target>
        <quality-of-service>DUPLICATES_OK</quality-of-service>
        <failure-retry-interval>500</failure-retry-interval>
        <max-retries>1</max-retries>
        <max-batch-size>500</max-batch-size>
        <max-batch-time>500</max-batch-time>
        <add-messageID-in-header>true</add-messageID-in-header>
    </jms-bridge>
</subsystem>

```

Im Beispiel oben sind die JNDI-Properties im **context**-Element für die **source** definiert. Wird das **context**-Element weggelassen, wie im **target**-Beispiel oben, so wird in der lokalen Instanz nach JMS-Ressourcen gesucht.

### 3.2.7.4. Migration Ihrer Applikation zur Verwendung von HornetQ als dem JMS-Provider

JBoss Messaging ist in der JBoss EAP 6 nicht mehr enthalten. Falls Ihre Applikation JBoss Messaging als Messaging Provider verwendet, so müssen Sie den JBoss Messaging Code durch HornetQ ersetzen.

#### Prozedur 3.19. Ehe Sie starten

1. Fahren Sie den Client und den Server herunter.
2. Erstellen Sie ein Backup sämtlicher JBoss Messaging Daten. Die Nachrichtendaten befinden sich in der Datenbank in Tabellen, denen ein **JBM\_** vorangestellt ist.

#### Prozedur 3.20. Ändern Sie Ihren Provider zu HornetQ

##### 1. Übertragen Sie die Konfigurationen

Übertragen Sie die bestehenden JBoss Messaging Konfigurationen auf die JBoss EAP Konfiguration. Die folgenden Konfigurationen befinden sich in Deployment-Deskriptoren auf dem JBoss Messaging Server:

- Connection Factories Dienstkonfiguration

Diese Konfiguration beschreibt die JMS-Connection-Factories, die mit dem JBoss Messaging Server deployt werden. JBoss Messaging konfiguriert Connection Factories in einer Datei namens **connection-factories-service.xml**, die sich im Deployment-Verzeichnis des Applikationsservers befindet.

- Zielkonfiguration

Diese Konfiguration beschreibt mit dem JBoss Messaging Server deployte JMS-Warteschlangen und Topics. Standardmäßig konfiguriert JBoss Messaging Destinationen in einer Datei namens **destinations-service.xml**, die sich im Deployment-Verzeichnis des Applikationsservers befindet.

- Message Bridge Dienstkonfiguration

Diese Konfiguration beinhaltet Bridge-Dienste, die mit dem JBoss Messaging Server deployt werden. Es werden keine Bridges standardmäßig deployt, so dass der Name der Deployment-Datei je nach Ihrer JBoss Messaging Installation variiert.

##### 2. Bearbeitung Ihres Applikationscodes

Falls der Applikationscode Standard-JMS verwendet, so sind keine Codeänderungen erforderlich. Falls sich die Applikation jedoch mit einem Cluster verbindet, so müssen Sie die HornetQ Dokumentation sorgfältig auf Clustering-Semantik prüfen. Clustering liegt außerhalb des Rahmens der JMS-Spezifikation und HornetQ und JBoss Messaging haben eine maßgeblich unterschiedliche Vorgehensweise in ihren jeweiligen Implementierungen der Clustering Funktionalität.

Falls die Applikation jedoch Features verwendet, die spezifisch für JBoss Messaging sind, so müssen Sie den Code so bearbeiten, dass er die äquivalenten Features in HornetQ nutzt.

Für weitere Informationen zur Konfiguration von Messaging mit HornetQ siehe [Abschnitt 3.2.7.5, »Konfigurieren Nachrichten-Vermittlung für HornetQ«](#)

### 3. Migration bestehender Nachrichten

Verschieben Sie alle Nachrichten in der JBoss Messaging Datenbank in das HornetQ Journal mittels einer JMS-Bridge. Eine Anleitung zur Konfiguration der JMS-Bridge finden Sie hier: [Abschnitt 3.2.7.2, »Konfiguration einer JMS-Bridge zur Migration bestehender JMS-Nachrichten zur JBoss EAP 6«](#).

[Fehler melden](#)

#### 3.2.7.5. Konfigurieren Nachrichten-Vermittlung für HornetQ

Die empfohlene Methode zum Konfigurieren der Nachrichten-Vermittlung in JBoss EAP 6 ist entweder mit der Management Konsole oder mit dem Management CLI. Sie können dauerhafte Änderungen mit jedem dieser Management-Tools machen, ohne dass Sie die **standalone.xml** oder **domain.xml** Konfigurations-Dateien manuell bearbeiten. Es ist allerdings nützlich sich mit den Komponenten der Nachrichten-Vermittlung der Standard Konfigurations-Dateien vertraut zu machen, wo Dokumentations-Beispiele, die Management-Tools verwenden, Ausschnitte der Konfigurations-Datei als Referenz angeben.

[Fehler melden](#)

### 3.2.8. Clustering-Änderungen

#### 3.2.8.1. Durchführung von Änderungen an Ihrer Applikation für Clustering

##### 1. Starten Sie die JBoss EAP 6 mit aktiviertem Clustering

Um Clustering in der JBoss EAP 5.x zu aktivieren müssen Sie Ihre Serverinstanzen mittels des **all**-Profils oder einer von dessen Ableitungen wie folgt starten:

```
$ EAP5_HOME/bin/run.sh -c all
```

In der JBoss EAP 6 hängt die Methode zur Aktivierung des Clustering davon ab, ob die Servers standalone sind oder in einer Managed Domain laufen.

##### a. Aktivierung von Clustering für in einer Managed Domain laufenden Servern

Um Clustering für mittels des Domain Controllers gestarteten Servern zu aktivieren, aktualisieren Sie Ihre **domain.xml** und designieren Sie eine Servergruppe zur Verwendung des **ha**-Profils und der **ha-sockets**-Socket-Binding-Gruppe. Zum Beispiel:

```
<server-groups>
  <server-group name="main-server-group" profile="ha">
    <jvm name="default">
      <heap size="64m" max-size="512m"/>
    </jvm>
    <socket-binding-group ref="ha-sockets"/>
  </server-group>
</server-group>
```

##### b. Aktivierung von Clustering für Standalone Server

Zur Aktivierung von Clustering für Standalone Server starten Sie den Server unter Verwendung der passenden Konfigurationsdatei wie folgt:

```
$ EAP_HOME/bin/standalone.sh --server-config=standalone-ha.xml -
Djboss.node.name=UNIQUE_NODE_NAME
```



## 2. Spezifizieren Sie die Bind-Adresse

Bei der JBoss EAP 5.x würden Sie normalerweise die für das Clustering zu verwendende Bind-Adresse unter Verwendung des **-b**-Befehlszeilenarguments wie folgt angeben:

```
$ EAP5_HOME/bin/run.sh -c all -b 192.168.0.2
```

JBoss EAP 6 bindet Sockets an die IP-Adressen und Interfaces, die in den **<interfaces>** Elementen in **standalone.xml**, **domain.xml** und **host.xml** Dateien enthalten sind. Die mit der JBoss EAP gelieferten Standardkonfigurationen umfassen zwei Interface-Konfigurationen:

```
<interfaces>
  <interface name="management">
    <inet-address
value="{jboss.bind.address.management:127.0.0.1}"/>
    </interface>
  <interface name="public">
    <inet-address value="{jboss.bind.address:127.0.0.1}"/>
    </interface>
</interfaces>
```

Diese Interface-Konfigurationen benutzen die Werte der System-Properties **jboss.bind.address.management** und **jboss.bind.address**. Wenn diese System-Properties nicht gesetzt wurden, wird für jeden Wert der Standardwert **127.0.0.1** benutzt.

Sie können auch die Bind-Adresse bei Serverstart als Befehlszeilenargument angeben, oder sie explizit innerhalb der JBoss EAP 6 Serverkonfigurationsdatei definieren.

- Bind-Argument auf der Befehlszeile bei Start des JBoss EAP Standalone Servers angeben.

Es folgt ein Beispiel für die Angabe einer Bind-Adresse in der Befehlszeile für einen Standalone Server:

```
EAP_HOME/bin/standalone.sh -Djboss.bind.address=127.0.0.1
```



### ANMERKUNG

Sie können auch das **-b** Argument benutzen, eine Abkürzung für **-Djboss.bind.address=127.0.0.1**:

```
EAP_HOME/bin/standalone.sh -b=127.0.0.1
```

Das JBoss EAP 5 Syntaxformat wird ebenfalls noch unterstützt:

```
EAP_HOME/bin/standalone.sh -b 127.0.0.1
```

Beachten Sie, dass das **-b** Argument nur das **public** Interface ändert. Es beeinflusst nicht das **management** Interface.

- Spezifizieren Sie die Bind-Adresse in der Server-Konfigurationsdatei.

Geben Sie die Bind-Adressen für Server, die in einer Managed Domain laufen, in der **domain/configuration/host.xml** Datei an. Für Standalone Server geben Sie die Bind-Adressen in der **standalone-ha.xml** Datei an.



Im folgenden Beispiel ist das **public**-Interface als das Standard-Interface für alle Sockets innerhalb der **ha-sockets**-Socket-Binding-Gruppe festgelegt.

```
<interfaces>
  <interface name="management">
    <inet-address value="192.168.0.2"/>
  </interface>
  <interface name="public">
    <inet-address value="192.168.0.2"/>
  </interface>
</interfaces>

<socket-binding-groups>
  <socket-binding-group name="ha-sockets" default-
interface="public">
    <!-- ... -->
  </socket-binding-group>
</socket-binding-groups>
```



#### ANMERKUNG

Wenn Sie die Bind-Adressen als hartcodierten Wert anstatt als System Property in der Konfigurationsdatei angeben, können Sie sie nicht mit einem Befehlszeilenargument außer Kraft setzen.

### 3. Konfiguration von **jvmRoute** zur Unterstützung von **mod\_jk** und **mod\_proxy**

Bei der JBoss EAP 5 wurde der Webserver **jvmRoute** mittels einer Property in der **server.xml**-Datei konfiguriert. Bei der JBoss Enterprise Application Platform 6 wird das **jvmRoute**-Attribut im Web-Untersystem der Serverkonfigurationsdatei unter Verwendung des **instance-id**-Attributs wie folgt konfiguriert:

```
<subsystem xmlns="urn:jboss:domain:web:1.1" default-virtual-
server="default-host" native="false" instance-id="
{JVM_ROUTE_SERVER}">
```

Das **{JVM\_ROUTE\_SERVER}** oben sollte durch die jvmRoute Server-ID ersetzt werden.

Die **instance-id** kann auch mittels der Management-Konsole eingestellt werden.

### 4. Festlegung von Multicast-Adresse und Port

Bei der JBoss EAP 5.x konnten Sie die für intra-Cluster verwendete Multicast-Adresse und den Port mittels der Befehlszeilenargumente **-u** und **-m** wie folgt festlegen:

```
$ EAP5_HOME/bin/run.sh -c all -u 228.11.11.11 -m 45688
```

Bei der JBoss EAP 6 sind die für intra-Cluster-Kommunikation verwendete Multicast-Adresse und der Port mittels des von dem relevanten JGroups Protokoll-Stacks referenzierten Socket-Binding wie folgt definiert:

```
<subsystem xmlns="urn:jboss:domain:jgroups:1.0" default-stack="udp">
  <stack name="udp">
    <transport type="UDP" socket-binding="jgroups-udp"/>
  </stack>
</subsystem>
```

```

        <!-- ... -->
    </stack>
</subsystem>

```

```

<socket-binding-groups>
  <socket-binding-group name="ha-sockets" default-
interface="public">
    <!-- ... -->
    <socket-binding name="jgroups-udp" port="55200" multicast-
address="228.11.11.11" multicast-port="45688"/>
    <!-- ... -->
  </socket-binding-group>
</socket-binding-groups>

```

Falls Sie die Multicast-Adresse und den Port lieber in der Befehlszeile festlegen möchten, so können Sie die Multicast-Adresse und Ports als System-Properties definieren und diese Properties an der Befehlszeile verwenden, wenn Sie den Server starten. Im folgenden Beispiel ist **jboss.mcast.addr** der Variablenname für die Multicast-Adresse und **jboss.mcast.port** ist der Variablenname für den Port.

```

<socket-binding name="jgroups-udp" port="55200"
multicast-address="${jboss.mcast.addr:230.0.0.4}" multicast-
port="${jboss.mcast.port:45688}"/>

```

Sie können dann Ihren Server mittels folgender Befehlszeilenargumente starten:

```

$ EAP_HOME/bin/domain.sh -Djboss.mcast.addr=228.11.11.11 -
Djboss.mcast.port=45688

```

## 5. Verwendung eines anderen Protocol-Stacks

Bei der JBoss EAP 5.x konnten Sie das standardmäßige Protocol-Stack, das für alle die **jboss.default.jgroups.stack** System-Property verwendenden Clustering-Dienst verwendet wurde, bearbeiten.

```

$ EAP5_HOME/bin/run.sh -c all -Djboss.default.jgroups.stack=tcp

```

Bei der JBoss EAP 6 ist das standardmäßige Protocol-Stack durch das JGroups-Subsystem innerhalb der **domain.xml** oder **standalone-ha.xml** definiert.

```

<subsystem xmlns="urn:jboss:domain:jgroups:1.0" default-stack="udp">
  <stack name="udp">
    <!-- ... -->
  </stack>
</subsystem>

```

## 6. Buddy-Replikation ersetzen

JBoss EAP 5.x benutzte JBoss Cache Buddy Replication um die Replikation von Daten zu allen Instanzen eines Clusters zu unterdrücken.

Bei der JBoss EAP 6 wurde Buddy Replication mit dem verteilten Cache von Infinispan, auch als **DIST** Modus bekannt, ersetzt. Verteilung ist ein starker Clusteringmodus, der es Infinispan ermöglicht linear zu skalieren, wenn dem Cluster mehr Server hinzugefügt werden. Es folgt ein Beispiel für die Konfiguration des Servers zur Verwendung des DIST Cachingmodus.

- a. Öffnen Sie eine Befehlszeile und starten Sie den Server mit HA oder Full Profile, zum Beispiel:

```
EAP_HOME/bin/standalone.sh -c standalone-ha.xml
```

- b. Öffnen Sie eine weitere Befehlszeile und verbinden Sie sich mit dem Management-CLI.

- In Linux geben Sie folgendes in der Befehlszeile ein:

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

- In Windows geben Sie folgendes in der Befehlszeile ein:

```
C:\>EAP_HOME\bin\jboss-cli.bat --connect
```

Sie sollten die folgende Antwort sehen:

```
Connected to standalone controller at localhost:9999
```

- c. Geben Sie die folgenden Befehle ein:

```
/subsystem=infinispan/cache-container=web/:write-  
attribute(name=default-cache,value=dist)  
/subsystem=infinispan/cache-container=web/distributed-  
cache=dist/:write-attribute(name=owners,value=3)  
:reload
```

Sie sollten die folgende Antwort nach jedem Befehl sehen:

```
"outcome" => "success"
```

Diese Befehle ändern das **dist** **<distributed-cache>**-Element in der **web** **<cache-container>**-Konfiguration im **infinispan**-Subsystem der **standalone-ha.xml**-Datei wie folgt:

```
<cache-container name="web" aliases="standard-session-cache"  
default-cache="dist"  
module="org.jboss.as.clustering.web.infinispan">  
  <transport lock-timeout="60000"/>  
  <replicated-cache name="repl" mode="ASYNC" batching="true">  
    <file-store/>  
  </replicated-cache>  
  <replicated-cache name="sso" mode="SYNC" batching="true"/>  
  <distributed-cache name="dist" owners="3" l1-lifespan="0"  
mode="ASYNC" batching="true">  
    <file-store/>  
  </distributed-cache>  
</cache-container>
```

Weitere Informationen finden Sie im Kapitel *Clustering in Web Applications* im *Development Guide* für die JBoss EAP 6 im Kundenportal unter [https://access.redhat.com/site/documentation/JBoss\\_Enterprise\\_Application\\_Platform/](https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/).

[Fehler melden](#)

### 3.2.8.2. Implementierung eines HA-Singleton

#### Zusammenfassung

Die folgende Vorgehensweise demonstriert, wie man einen Dienst deployt, der im SingletonServer decorator eingebunden ist und als Cluster-weiter Singleton Dienst benutzt wird. Dieser Dienst aktiviert einen festgelegten Timer, der nur einmal im Cluster gestartet wird.

#### Prozedur 3.21. Implementierung eines HA-Singleton-Dienstes

##### 1. Schreiben Sie die HA-Singleton-Dienst-Applikation.

Nachfolgend sehen Sie ein einfaches Beispiel für einen **Service**, der mit dem **SingletonService** Decorator eingebunden ist um als Singleton Dienst deployt zu werden. Ein vollständiges Beispiel finden Sie im **cluster-ha-singleton** Quickstart, der zusammen mit Red Hat JBoss Enterprise Application Platform 6 geliefert wird. Dieser Quickstart enthält sämtliche Anweisungen um die Applikation zu erstellen und zu deployen.

##### a. Erstellen Sie einen Dienst.

Nachfolgend sehen Sie ein Beispiel für einen Dienst:

```
package org.jboss.as.quickstarts.cluster.hasingleton.service.ejb;

import java.util.Date;
import java.util.concurrent.atomic.AtomicBoolean;

import javax.naming.InitialContext;
import javax.naming.NamingException;

import org.jboss.logging.Logger;
import org.jboss.msc.service.Service;
import org.jboss.msc.service.ServiceName;
import org.jboss.msc.service.StartContext;
import org.jboss.msc.service.StartException;
import org.jboss.msc.service.StopContext;

/**
 * @author <a href="mailto:wfink@redhat.com">Wolf-Dieter Fink</a>
 */
public class HATimerService implements Service<String> {
    private static final Logger LOGGER =
        Logger.getLogger(HATimerService.class);
    public static final ServiceName SINGLETON_SERVICE_NAME =
        ServiceName.JBOSS.append("quickstart", "ha", "singleton",
            "timer");

    /**
     * A flag whether the service is started.
     */
    private final AtomicBoolean started = new
        AtomicBoolean(false);

    /**
     * @return the name of the server node
```

```

        */
        public String getValue() throws IllegalStateException,
        IllegalArgumentException {
            LOGGER.infof("%s is %s at %s",
            HATimerService.class.getSimpleName(), (started.get() ? "started"
            : "not started"), System.getProperty("jboss.node.name"));
            return "";
        }

        public void start(StartContext arg0) throws StartException {
            if (!started.compareAndSet(false, true)) {
                throw new StartException("The service is still
started!");
            }
            LOGGER.info("Start HASingleton timer service '" +
this.getClass().getName() + "'");

            final String node =
System.getProperty("jboss.node.name");
            try {
                InitialContext ic = new InitialContext();
                ((Scheduler) ic.lookup("global/jboss-cluster-ha-
singleton-
service/SchedulerBean!org.jboss.as.quickstarts.cluster.hasingleto
n.service.ejb.Scheduler")).initialize("HASingleton timer @" +
node + " " + new Date());
            } catch (NamingException e) {
                throw new StartException("Could not initialize
timer", e);
            }
        }

        public void stop(StopContext arg0) {
            if (!started.compareAndSet(true, false)) {
                LOGGER.warn("The service '" +
this.getClass().getName() + "' is not active!");
            } else {
                LOGGER.info("Stop HASingleton timer service '" +
this.getClass().getName() + "'");
                try {
                    InitialContext ic = new InitialContext();
                    ((Scheduler) ic.lookup("global/jboss-cluster-ha-
singleton-
service/SchedulerBean!org.jboss.as.quickstarts.cluster.hasingleto
n.service.ejb.Scheduler")).stop();
                } catch (NamingException e) {
                    LOGGER.error("Could not stop timer", e);
                }
            }
        }
    }
}

```

b. **Erstellen Sie einen Aktivator, der den Service als geclustertes Singleton installiert.**

Die folgende Liste ist ein Beispiel für einen Service Aktivator, der den **HATimerService** als geclusterten Singleton Dienst installiert:

```

package org.jboss.as.quickstarts.cluster.hasingleton.service.ejb;

import org.jboss.as.clustering.singleton.SingletonService;
import org.jboss.logging.Logger;
import org.jboss.msc.service.DelegatingServiceContainer;
import org.jboss.msc.service.ServiceActivator;
import org.jboss.msc.service.ServiceActivatorContext;
import org.jboss.msc.service.ServiceController;

/**
 * Service activator that installs the HATimerService as a
 * clustered singleton service
 * during deployment.
 *
 * @author Paul Ferraro
 */
public class HATimerServiceActivator implements ServiceActivator
{
    private final Logger log = Logger.getLogger(this.getClass());

    @Override
    public void activate(ServiceActivatorContext context) {
        log.info("HATimerService will be installed!");

        HATimerService service = new HATimerService();
        SingletonService<String> singleton = new
SingletonService<String>(service,
HATimerService.SINGLETON_SERVICE_NAME);
        /*
         * To pass a chain of election policies to the singleton,
         for example,
         * to tell JGroups to prefer running the singleton on a
         node with a
         * particular name, uncomment the following line:
         */
        // singleton.setElectionPolicy(new
PreferredSingletonElectionPolicy(new
SimpleSingletonElectionPolicy(), new
NamePreference("node2/cluster")));

        singleton.build(new
DelegatingServiceContainer(context.getServiceTarget(),
context.getServiceRegistry()))
            .setInitialMode(ServiceController.Mode.ACTIVE)
            .install()
        ;
    }
}

```



## ANMERKUNG

Das oben angegebene Codebeispiel benutzt eine Klasse, **org.jboss.as.clustering.singleton.SingletonService**, die Teil des JBoss EAP private APIs ist. Ein öffentliches API wird in der EAP 7 Release verfügbar sein und die private Klasse wird überholt sein, aber diese Klassen werden für die Dauer des EAP 6.x Releasezyklus erhalten und verfügbar bleiben.

### c. Erstellen einer ServiceActivator Datei

Erstellen Sie eine Datei namens **org.jboss.msc.service.ServiceActivator** im **resources/META-INF/services/** Verzeichnis der Applikation. Fügen Sie eine Zeile hinzu, die den im vorherigen Schritt erstellten, vollqualifizierten Namen der ServiceActivator Klasse enthält.

```
org.jboss.as.quickstarts.cluster.hasingleton.service.ejb.HATimerServiceActivator
```

### d. Erstellen Sie eine Singleton Bean, die einen Timer implementiert, der als Cluster-weiter Singleton Timer dient.

Diese Singleton Bean darf kein Remote Interface haben und Sie dürfen keine Verweise auf sein lokales Interface aus anderen EJB irgendwelcher Applikationen anbringen. Dies verhindert einen Lookup durch einen Client oder andere Komponenten und stellt sicher, dass der SingletonService vollkommene Kontrolle über das Singleton hat.

#### i. Erstellen des Scheduler Interface

```
package
org.jboss.as.quickstarts.cluster.hasingleton.service.ejb;

/**
 * @author <a href="mailto:wfink@redhat.com">Wolf-Dieter
 * Fink</a>
 */
public interface Scheduler {

    void initialize(String info);

    void stop();

}
```

#### ii. Erstellen Sie die Singleton Bean, die den Cluster-weiten Singleton Timer implementiert.

```
package
org.jboss.as.quickstarts.cluster.hasingleton.service.ejb;

import javax.annotation.Resource;
import javax.ejb.ScheduleExpression;
import javax.ejb.Singleton;
import javax.ejb.Timeout;
import javax.ejb.Timer;
import javax.ejb.TimerConfig;
```

```

import javax.ejb.TimerService;

import org.jboss.logging.Logger;

/**
 * A simple example to demonstrate a implementation of a
 * cluster-wide singleton timer.
 *
 * @author <a href="mailto:wfink@redhat.com">Wolf-Dieter
 * Fink</a>
 */
@Singleton
public class SchedulerBean implements Scheduler {
    private static Logger LOGGER =
        Logger.getLogger(SchedulerBean.class);
    @Resource
    private TimerService timerService;

    @Timeout
    public void scheduler(Timer timer) {
        LOGGER.info("HASingletonTimer: Info=" +
            timer.getInfo());
    }

    @Override
    public void initialize(String info) {
        ScheduleExpression sexpr = new ScheduleExpression();
        // set schedule to every 10 seconds for demonstration
        sexpr.hour("*").minute("*").second("0/10");
        // persistent must be false because the timer is
        // started by the HASingleton service
        timerService.createCalendarTimer(sexpr, new
            TimerConfig(info, false));
    }

    @Override
    public void stop() {
        LOGGER.info("Stop all existing HASingleton timers");
        for (Timer timer : timerService.getTimers()) {
            LOGGER.trace("Stop HASingleton timer: " +
                timer.getInfo());
            timer.cancel();
        }
    }
}

```

## 2. Starten Sie jede JBoss EAP 6 Instanz mit aktiviertem Clustering.

Um Clustering für Standalone Server zu aktivieren, müssen Sie jeden Server mit dem **HA** Profil starten, indem Sie einen eindeutigen Knotennamen und Port Offset für jede Instanz benutzen.

- Benutzen Sie in Linux folgende Befehlssyntax um die Server zu starten:



```
EAP_HOME/bin/standalone.sh --server-config=standalone-ha.xml -
Djboss.node.name=UNIQUE_NODE_NAME -Djboss.socket.binding.port-
offset=PORT_OFFSET
```

### Beispiel 3.3. Start mehrerer Standalone Server in Linux

```
$ EAP_HOME/bin/standalone.sh --server-config=standalone-ha.xml
-Djboss.node.name=node1
$ EAP_HOME/bin/standalone.sh --server-config=standalone-ha.xml
-Djboss.node.name=node2 -Djboss.socket.binding.port-offset=100
```

- Benutzen Sie in Microsoft Windows folgende Befehlssyntax um den Server zu starten:

```
EAP_HOME\bin\standalone.bat --server-config=standalone-ha.xml -
Djboss.node.name=UNIQUE_NODE_NAME -Djboss.socket.binding.port-
offset=PORT_OFFSET
```

### Beispiel 3.4. Start mehrerer Standalone Server in Microsoft Windows

```
C:> EAP_HOME\bin\standalone.bat --server-config=standalone-
ha.xml -Djboss.node.name=node1
C:> EAP_HOME\bin\standalone.bat --server-config=standalone-
ha.xml -Djboss.node.name=node2 -Djboss.socket.binding.port-
offset=100
```



#### ANMERKUNG

Wenn Sie die Befehlszeilenargumente lieber nicht benutzen wollen, können Sie die **standalone-ha.xml** Datei für jede Serverinstanz darauf konfigurieren, auf einem separaten Interface zu binden.

### 3. Deployment der Applikation auf die Server

Folgender Maven Befehl deployt die Applikation auf einem Standalone Server, der auf Standardports läuft.

```
mvn clean install jboss-as:deploy
```

Übertragen Sie die Servernamen um zusätzliche Server zu deployen. Wenn es auf einem anderen Host ist, übertragen Sie Hostnamen und Portnummer auf die Befehlszeile:

```
mvn clean package jboss-as:deploy -Djboss-as.hostname=localhost -
Djboss-as.port=10099
```

Siehe **cluster-ha-singleton** Quickstart, welches mit JBoss EAP 6 für Maven Konfiguration und Deployment Details geliefert wird.

### 3.2.9. Änderungen beim Deployment im Dienst-Stil

#### 3.2.9.1. Aktualisierung von Applikationen, die Deployments im Dienst-Stil verwenden

##### Zusammenfassung

MBeans waren Teil der Kernarchitektur in früheren Versionen der Red Hat JBoss Enterprise Application Platform. JBoss Service Archive (SAR) Deployments, die JBoss spezifische **jboss-service.xml** und **jboss-beans.xml** Deskriptoren im Dienst-Stil benutzen, wurden vom Applikationsserver zur Erstellung von MBeans, basierend auf JBoss Beans, benutzt. Die interne Architektur wurde in JBoss EAP 6 geändert und basiert nicht mehr auf einer MBean JMX Architektur. MBeans sind nicht mehr Teil der Kernarchitektur. Sie sind jetzt Wrapper für das Management API.

Wenn Ihre Applikation Deployment Deskriptoren im Dienst-Stil benutzt, wird sie vielleicht weiterhin in JBoss EAP 6 laufen, so lange sie nur von MBeans abhängt, die Ihre Applikation definieren und nicht von JBoss Management API MBean Wrappers. In JBoss EAP 6 können SARs nur MBean Abhängigkeiten auf MBeans deklarieren, die von einem anderen SAR Deployment erstellt wurden. Das bedeutet, wenn Ihre Applikation von MBeans abhängt, die JBoss EAP erstellte, wie eine MBean für ein EJB oder eine Messagingkomponente, werden sie nicht mehr funktionieren. Die einzigen zuverlässigen MBeans sind andere MBeans, die Sie in anderen **jboss-service.xml** files definiert haben.

Das in früheren Versionen der JBoss EAP benutzte JBoss Service Archive (SAR) und Service-Style Deskriptoren sind nicht Teil der Java EE 6 Spezifikation und werden nicht für die Benutzung in JBoss EAP 6 empfohlen. Es wird empfohlen, dass Sie Ihre Applikation auf die Java EE 6 Spezifikation modifizieren. Für MBeans Singletons sollten Sie den Code auf die Verwendung des Java EE 6 **@Singleton** modifizieren. Weitere Informationen zu Erstellung und Deployment von MBean Diensten finden Sie in dem Kapitel *JBoss MBean Services* im *Development Guide* für JBoss Enterprise Application Platform 6 im Kundenportal unter

[https://access.redhat.com/documentation/JBoss\\_Enterprise\\_Application\\_Platform/](https://access.redhat.com/documentation/JBoss_Enterprise_Application_Platform/).

[Fehler melden](#)

### 3.2.10. Änderungen bei Remote-Aufrufen

#### 3.2.10.1. Migration von JBoss EAP 5 deployten Applikationen, die Remote-Aufrufe zur JBoss EAP 6 tätigen

##### Zusammenfassung

Bei der JBoss EAP 5 war das EJB-Remote-Interface standardmäßig in JNDI unter dem Namen "ejbName/local" für lokale Interfaces und "ejbName/remote" für Remote-Interfaces gebunden. Die Client-Applikation suchte das Bean dann mittels "ejbName/remote".

Bei der JBoss EAP 6 wurde ein neues, spezifisches EJB Client API zur Ausführung von Aufrufen eingeführt. Wenn Sie jedoch Ihren Code nicht neu schreiben möchten die neue API zu benutzen, so können Sie den vorhandenen Code ändern um **ejb:BEAN\_REFERENCE** Remote-Zugriff auf EJBs mit folgender Syntax zu benutzen.

Für Stateless Beans lautet die **ejb:BEAN\_REFERENCE**-Syntax:

```
ejb:<app-name>/<module-name>/<distinct-name>/<bean-name>!<fully-qualified-  
classname-of-the-remote-interface>
```

Für Stateful Beans lautet die **ejb:BEAN\_REFERENCE**-Syntax:

```
ejb:<app-name>/<module-name>/<distinct-name>/<bean-name>!<fully-qualified-
classname-of-the-remote-interface>?stateful
```

Die zu ersetzenden Werte in der obigen Syntax sind:

- **<app-name>** - der Applikationsname der deployten EJBs. Dies ist in der Regel der ear-Name ohne das .ear-Suffix, allerdings kann der Name in der application.xml-Datei außer Kraft gesetzt werden. Wird die Applikation nicht als ein .ear deployt, so ist dieser Wert ein leerer String. Nehmen Sie an, dass dieses Beispiel nicht als ein EAR deployt wurde.
- **<module-name>** - der Modulname der deployten EJBs auf dem Server. Dies ist in der Regel der jar-Name des EJB-Deployments ohne den .jar-Suffix, kann aber durch Verwendung der ejb-jar.xml außer Kraft gesetzt werden. Nehmen Sie in diesem Beispiel an, dass die EJBs in einem jboss-ejb-remote-app.jar deployt wurden, so dass der Modulname jboss-ejb-remote-app lautet.
- **<distinct-name>** - ein optionaler eindeutiger Name für das EJB. Dieses Beispiel verwendet keinen eindeutigen Namen und verwendet einen leeren String.
- **<bean-name>** - der Standard, der einfache Klassenname der Bean-Implementierungsklasse.
- **<fully-qualified-classname-of-the-remote-interface>** - der Remote-Ansicht vollqualifizierte Klassenname.

### Aktualisierung des Client-Code

Nehmen Sie an, Sie hätten das folgende stateless EJB auf einen JBoss EAP 6 Server deployt. Beachten Sie, dass es eine Remote-Ansicht für das Bean offenlegt.

```
@Stateless
@Remote(RemoteCalculator.class)
public class CalculatorBean implements RemoteCalculator {

    @Override
    public int add(int a, int b) {
        return a + b;
    }

    @Override
    public int subtract(int a, int b) {
        return a - b;
    }
}
```

Bei der JBoss EAP 5 war der Client EJB-Lookup und Aufruf ähnlich wie folgt kodiert:

```
InitialContext ctx = new InitialContext();
RemoteCalculator calculator = (RemoteCalculator)
ctx.lookup("CalculatorBean/remote");
int a = 204;
int b = 340;
int sum = calculator.add(a, b);
```

Bei der JBoss EAP 6 wird der Client EJB-Lookup und Aufruf mittels der obigen Informationen wie folgt kodiert:

```
final Hashtable jndiProperties = new Hashtable();
```

```

jndiProperties.put(Context.URL_PKG_PREFIXES,
"org.jboss.ejb.client.naming");
final Context context = new InitialContext(jndiProperties);
final String appName = "";
final String moduleName = "jboss-ejb-remote-app";
final String distinctName = "";
final String beanName = CalculatorBean.class.getSimpleName();
final String viewClassName = RemoteCalculator.class.getName();
final RemoteCalculator statelessRemoteCalculator = (RemoteCalculator)
context.lookup("ejb:" + appName + "/" + moduleName + "/" + distinctName +
"/" + beanName + "!" + viewClassName);

int a = 204;
int b = 340;
int sum = statelessRemoteCalculator.add(a, b);

```

Falls Ihr Client auf ein stateful EJB zugreift, so müssen Sie "?stateful" wie folgt an das Ende des Kontext-Lookup anhängen:

```

final RemoteCalculator statefulRemoteCalculator = (RemoteCalculator)
context.lookup("ejb:" + appName + "/" + moduleName + "/" + distinctName +
"/" + beanName + "!" + viewClassName + "?stateful")

```

Ein komplettes Arbeitsbeispiel samt Server- und Clientcode finden Sie in den Quickstarts. Weitere Informationen finden Sie unter *Review the Quickstart Tutorials* im Kapitel *Get Started Developing Applications* im *Entwicklungshandbuch* für die JBoss EAP 6 unter [https://access.redhat.com/site/documentation/JBoss\\_Enterprise\\_Application\\_Platform/](https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/).

Weitere Informationen zu Remote-Aufrufen unter Verwendung von JNDI finden Sie unter [Abschnitt 3.2.10.2, »Remote-Aufruf eines Session-Beans mittels JNDI«](#).

[Fehler melden](#)

### 3.2.10.2. Remote-Aufruf eines Session-Beans mittels JNDI

Diese Aufgabe beschreibt, wie Support einem Remote-Client für den Aufruf von Session-Beans mittels JNDI hinzugefügt wird. Die Aufgabe geht davon aus, dass das Projekt mittels Maven erstellt wird.

Der **ejb-remote**-Quickstart enthält Maven-Projekte, die diese Funktionalität demonstrieren. Der Quickstart enthält Projekte für das Deployment beider Session-Beans und den Remote-Client. Die Code-Beispiele unten wurden dem Remote-Client-Projekt entnommen.

Diese Aufgabe geht davon aus, dass die Session-Beans keine Authentifizierung benötigen.



#### WARNUNG

Red Hat empfiehlt, dass Sie SSL zugunsten von TLSv1.1 oder TLSv1.2 in allen betroffenen Paketen explizit deaktivieren.

## Voraussetzungen

Ehe Sie anfangen, müssen folgende Voraussetzungen erfüllt sein:

- Sie müssen bereits ein einsatzbereites Maven-Projekt besitzen.
- Die Konfiguration für das JBoss EAP 6 Maven Repository wurde bereits hinzugefügt.
- Die Session-Beans, die Sie aufrufen möchten, sind bereits deployt.
- Die deployten Session-Beans implementieren Remote Business-Interfaces.
- Die Remote Business-Interfaces der Session-Beans sind als eine Maven-Abhängigkeit verfügbar. Falls die Remote Business-Interfaces nur als eine JAR-Datei verfügbar sind, so empfiehlt es sich das JAR als Baustein Ihrem Maven-Repository hinzuzufügen. In der Maven-Dokumentation finden Sie im **install:install-file** Ziel weitere Informationen <http://maven.apache.org/plugins/maven-install-plugin/usage.html>.
- Sie müssen den Host-Namen und den JNDI-Port des Servers kennen, der die Session-Beans hostet.

Um ein Session-Bean von einem Remote Client aufzurufen, müssen Sie das Projekt zuerst ordnungsgemäß konfigurieren.

### Prozedur 3.22. Hinzufügen der Maven Projektkonfiguration für Remote-Aufrufe von Session Beans

#### 1. Fügen Sie die erforderlichen Projektabhängigkeiten hinzu

Die **pom.xml** für das Projekt muss aktualisiert werden, um die notwendigen Abhängigkeiten zu enthalten.

#### 2. Fügen Sie die **jboss-ejb-client.properties**-Datei hinzu

Das JBoss EJB Client-API erwartet eine Datei im root des Projekts namens **jboss-ejb-client.properties**, die die Verbindungsinformationen für den JNDI-Dienst enthält. Fügen Sie diese Datei dem **src/main/resources/-**Verzeichnis Ihres Projekts mit folgendem Inhalt hinzu.

```
# In the following line, set SSL_ENABLED to true for SSL
remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABLED=false
remote.connections=default
# Uncomment the following line to set SSL_STARTTLS to true for SSL
#
remote.connection.default.connect.options.org.xnio.Options.SSL_STARTTLS=true
remote.connection.default.host=localhost
remote.connection.default.port = 4447
remote.connection.default.connect.options.org.xnio.Options.SASL_POLICY_NOANONYMOUS=false
# Add any of the following SASL options if required
#
remote.connection.default.connect.options.org.xnio.Options.SASL_POLICY_NOANONYMOUS=false
#
remote.connection.default.connect.options.org.xnio.Options.SASL_POLICY_NOPLAINTEXT=false
```

```
#
remote.connection.default.connect.options.org.xnio.Options.SASL_DISA
LLOWED_MECHANISMS=JBoss-LOCAL-USER
```

Ändern Sie den Host-Namen und Port so, dass diese mit Ihrem Server übereinstimmen. **4447** ist die Standardportnummer. Für eine sichere Verbindung setzen Sie die **SSL\_ENABLED**-Zeile auf **true** und kommentieren Sie die **SSL\_STARTTLS**-Zeile aus. Das Remoting-Interface im Container unterstützt gesicherte und ungesicherte Verbindungen am selben Port.

### 3. Fügen Sie Abhängigkeiten für die Remote Business Interfaces hinzu

Fügen Sie die Maven-Abhängigkeiten der **pom.xml** für die Remote Business-Interfaces der Session-Beans hinzu.

```
<dependency>
  <groupId>org.jboss.as.quickstarts</groupId>
  <artifactId>jboss-ejb-remote-server-side</artifactId>
  <type>ejb-client</type>
  <version>${project.version}</version>
</dependency>
```

Jetzt, da das Projekt ordnungsgemäß konfiguriert ist, können Sie den Code für den Zugriff und Aufruf der Session-Beans hinzufügen.

## Prozedur 3.23. Erhalt eines Bean-Proxy mittels JNDI und Methodenaufruf des Bean

### 1. Handhabung geprüfter Ausnahmen

Zwei der im folgenden Code (**InitialContext()** und **lookup()**) verwendeten Methoden, besitzen eine geprüfte Ausnahme vom Typ **javax.naming.NamingException**. Diese Methodenaufrufe müssen entweder in einem "try/catch"-Block eingeschlossen sein, der die **NamingException** auffängt oder in einer Methode, die zur Meldung der **NamingException** deklariert ist. Der **ejb-remote**-Quickstart verwendet die zweite Variante.

### 2. Erstellen Sie einen JNDI-Kontext

Ein JNDI-Kontextobjekt liefert den Mechanismus zum Anfragen von Ressourcen vom Server. Erstellen Sie einen JNDI-Kontext mittels folgendem Code:

```
final Hashtable jndiProperties = new Hashtable();
jndiProperties.put(Context.URL_PKG_PREFIXES,
"org.jboss.ejb.client.naming");
final Context context = new InitialContext(jndiProperties);
```

Die Verbindungseigenschaften für den JNDI-Dienst werden aus der **jboss-ejb-client.properties**-Datei gelesen.

### 3. Verwenden Sie die lookup() Methode des JNDI-Kontexts zum Erhalt eines Bean Proxy

Rufen Sie die **lookup()**-Methode des Bean-Proxy auf und geben sie dieser den JNDI-Namen des benötigten Session-Beans. Dies gibt ein Objekt wieder, das an den Typ des Remote Business-Interface verteilt werden muss, das die Methoden enthält, die Sie aufzurufen wünschen.

```
final RemoteCalculator statelessRemoteCalculator =
(RemoteCalculator) context.lookup(
```

```
"ejb:/jboss-ejb-remote-server-side//CalculatorBean!" +
RemoteCalculator.class.getName());
```

Session-Bean JNDI-Namen werden unter Verwendung einer besonderen Syntax definiert. Weitere Informationen finden Sie unter [Abschnitt 3.2.10.3, »EJB JNDI Namengebungsreferenz«](#).

#### 4. Aufrufen von Methoden

Jetzt da Sie ein Proxy-Bean-Objekt besitzen, können Sie jede beliebige der im Remote Business-Interface enthaltenen Methoden aufrufen.

```
int a = 204;
int b = 340;
System.out.println("Adding " + a + " and " + b + " via the remote
stateless calculator deployed on the server");
int sum = statelessRemoteCalculator.add(a, b);
System.out.println("Remote calculator returned sum = " + sum);
```

Das Proxy-Bean gibt die Methodenaufrufanfrage an das Session-Bean auf dem Server, wo es ausgeführt wird. Das Ergebnis wird an das Proxy-Bean wiedergegeben, das es dann an den Aufrufer wiedergibt. Die Kommunikation zwischen dem Proxy-Bean und dem Remote Session Bean ist für den Aufrufer transparent.

Sie sollten jetzt zur Konfiguration eines Maven-Projekts zur Unterstützung des Aufrufs von Session-Beans an einem Remote Server in der Lage sein und den Codeaufruf der Session-Beans-Methoden mittels eines vom Server mittels JNDI erhaltenen Proxy-Beans schreiben können.

[Fehler melden](#)

### 3.2.10.3. EJB JNDI Namengebungsreferenz

JNDI Lookup Name für eine Session Bean hat die Syntax:

```
ejb:<appName>/<moduleName>/<distinctName>/<beanName>!<viewClassName>?
stateful
```

#### <appName>

Wenn die JAR Datei der Session Bean innerhalb eines Enterprise Archivs (EAR) deployt wurde, dann ist es der Name dieses EAR. Standardmäßig ist der Name eines EAR sein Dateiname ohne das **.ear** Suffix. Der Applikationsname kann auch in seiner **application.xml** Datei außer Kraft gesetzt werden. Wenn die Session Bean nicht in einem EAR deployt wurde, lassen Sie es leer.

#### <moduleName>

Der Modulname ist der Name der JAR Datei, in der die Session Bean deployt ist. Standardmäßig ist der Name der JAR Datei ihr Dateiname ohne das **.jar** Suffix. Der Modulname kann auch in der **ejb-jar.xml** Datei des JARs außer Kraft gesetzt werden.

#### <distinctName>

Die JBoss EAP 6 ermöglicht jedem Deployment einen optionalen eindeutigen Namen zu bestimmen. Wenn das Deployment keinen eindeutigen Namen hat, lassen Sie es leer.

#### <beanName>



Der Bean Name ist der Klassenname der aufzurufenden Session Bean.

#### **<viewClassName>**

Der Ansichtsklassennamen ist der vollqualifizierte Klassenname des Remote Interface. Das beinhaltet den Paketnamen des Interface.

#### **?stateful**

Das **?stateful** Suffix ist erforderlich, wenn der JNDI Name auf eine statusbehaftete Session Bean verweist. Es ist bei anderen Bean Typen nicht eingeschlossen.

[Fehler melden](#)

## **3.2.11. EJB 2.x Änderungen**

### **3.2.11.1. Aktualisierung von Applikationen, die EJB 2.x verwenden**

JBoss EAP 6 wurde auf offenen Standards erstellt und ist kompatibel mit der Java Enterprise Edition 6 Spezifikation. Während der Applikationsserver Support für EJB 2.x bietet, bietet er möglicherweise keinen Support mehr für Features, die über die Spezifikation hinaus gehen. Bedenken Sie, dass die Java EE 7 Spezifikation EJB 2.x als optional markierte, demnach wird stark empfohlen, dass Sie Ihren Applikationscode in die EJB 3.x Spezifikation neu schreiben.

Wenn Sie dennoch Ihren EJB 2.x Code migrieren möchten, werden Sie in den meisten Fällen ein paar Änderungen vornehmen müssen, damit es in JBoss EAP 6 läuft. Dieses Thema beschreibt einige der Änderungen, die Sie vornehmen müssen, damit EJB 2.x auf JBoss EAP 6 läuft.

#### **Konfigurationsänderungen erforderlich um EJB 2.x auf JBoss EAP 6 auszuführen**

##### **Starten Sie den Server mit dem "Full Profile"**

EJB 2.x Container Managed Persistence (CMP) Beans benötigen das Java Enterprise Edition 6 Full Profil. Dieses Profil beinhaltet Konfigurationselemente, die zur Ausführung von CMP EJBs benötigt werden.

Dieses Konfigurationsprofil beinhaltet das **org.jboss.as.cmp** Erweiterungsmodul:

```
<extensions>
  ...
  <extension module="org.jboss.as.cmp"/>
  ...
</extensions>
```

Es beinhaltet auch das **cmp** Subsystem:

```
<profiles>
  ...
  <subsystem xmlns="urn:jboss:domain:cmp:1.1"/>
  ...
</profiles>
```



Um einen JBoss EAP 6 Standalone Server mit dem vollständigen Profil zu starten, übertragen Sie das **-c standalone-full.xml** oder **-c standalone-full-ha.xml** Argument auf die Befehlszeile, wenn Sie den Server starten.

### Containerkonfiguration wird nicht mehr unterstützt

In früheren Versionen der JBoss EAP war es möglich verschiedene Container für die CMP Entity und andere Beans zu konfigurieren und zu benutzen, indem man Referenzen in die **jboss.xml** Applikations-Deployment Deskriptordatei eingab. Es gab zum Beispiel generell verschiedene Konfigurationen für SLSB zu Session Beans.

Bei der JBoss EAP 6.x ist es möglich EJB 2 Entity Beans mit einem Standard-Container zu benutzen. Die verschiedenen Containerkonfigurationen werden jedoch nicht mehr unterstützt. Empfohlen wird der Ansatz, nach dem die EJB2 Stateful Session Beans (SFSB), Stateless Session Beans (SLSB), Message Driven Beans (MDB) nach EJB 3 zu migrieren und für die Container-Managed Persistence (CMP) und Bean-Managed Persistence (BMP) Entity BeansJava Persistence API (JPA) gemäß der EJB 3 Spezifikation zu benutzen.

Die standardmäßige Containerkonfiguration in JBoss EAP 6 beinhaltet einige Änderungen für EJB 2 CMP Beans:

- Pessimistisches Sperren ist standardmäßig aktiviert. Das kann zu Deadlocks führen.
- Der deadlock detection code, den es in der CMP Layer in JBoss EAP 5.x gab, gibt es in JBoss EAP 6 nicht mehr.

In JBoss EAP 5.x war es auch möglich Caching, Pooling, **commit-options** und den Interzeptor Stack anzupassen. In JBoss EAP 6 ist das nicht mehr möglich. Es gibt nur eine Implementation, die der **Instance Per Transaction** Policy mit **commit-option C** ähnlich ist. Wenn Sie eine Applikation migrieren, die die **cmp2.x jdbc2 pm** Entity Bean Container Konfiguration benutzt, welche einen CMP2.x kompatiblen, JDBC basierten Persistenz-Manager benutzt, wird es sich auf die Performance auswirken. Dieser Container wurde für die Performance optimiert. Es wird empfohlen, dass Sie diese Entities zu EJB 3 migrieren, bevor Sie die Applikation migrieren.

### Serverseitige Interzeptor-Konfiguration

JBoss EAP 6 unterstützt den Java EE **Interceptor** unter Verwendung der **@Interceptors** und **@AroundInvoke** Annotationen. Dies ermöglicht jedoch nicht die Bearbeitung außerhalb der Zugriffsrechte oder Transaktion.

In früheren Versionen der JBoss EAP war es möglich den Interzeptor Stack zu bearbeiten um benutzerdefinierte Interzeptoren für jeden EJB Aufruf zu haben. Dies wurde oft verwendet um benutzerdefinierte Zugriffsrechte zu implementieren oder Mechanismen vor Sicherheitsüberprüfungen, Transaktionsüberprüfungen oder Erstellung auszuprobieren. JBoss EAP 6.1 führte Container-Interzeptoren ein, um ähnliche Funktionalitäten zu bieten. Weitere Informationen über Container-Interzeptoren finden Sie in dem Kapitel *Container Interceptors* im *Development Guide* für JBoss EAP.

Ein anderer Ansatz für bessere Kontrolle vor, während oder nach der Commitphase einer Transaktion unter Einhaltung der Java EE Spezifikation ist es, das Transaction Synchronization Registry zu benutzen um einen Listener hinzuzufügen.

Die Ressource kann über eine der folgenden Methoden abgefragt werden:

- Benutzen Sie den **InitialContext**

```
TransactionSynchronizationRegistry tsr =
```

```
(TransactionSynchronizationRegistry)
    new
    InitialContext().lookup("java:jboss/TransactionSynchronizationRegi
stry");
    tsr.registerInterposedSynchronization(new MyTxCallback());
```

- Verwendung von Einspeisung

```
@Resource(mappedName =
"java:comp/TransactionSynchronizationRegistry")
TransactionSynchronizationRegistry tsr;
...
tsr.registerInterposedSynchronization(new MyTxCallback());
```

Die Callback Routine muss das **javax.transaction.Synchronization** Interface implementieren. Benutzen Sie die **beforeCompletion{}** Methode um Prüfungen durchzuführen, bevor die Transaktion ausgeführt oder zurückgesetzt wurde. Wenn eine **RuntimeException** von dieser Methode ausgelöst wird, wird die Transaktion zurückgesetzt und der Client wird mit einer **EJBTransactionRolledbackException** informiert. Im Falle einer XA-Transaction werden alle Ressourcen gemäß dem XA-Vertrag zurückgesetzt. Für Geschäftslogik aktivieren ist es auch möglich über die **afterCompletion(int txStatus)** Methode vom Transaktionsstatus abhängig zu sein. Wenn eine **RuntimeException** von dieser Methode ausgelöst wird, bleibt die Transaktion in ihrem früheren Status, entweder zurückgesetzt oder ausgeführt, und der Client wird nicht informiert. Nur der Transaktions-Manager zeigt eine Warnung innerhalb der Serverprotokolldateien an.

### Serverseitige Konfiguration für Clientseitige Interzeptoren

In früheren Versionen der JBoss EAP war es möglich, den Client-Interzeptor innerhalb der Serverkonfiguration zu konfigurieren und nur die Klassen mit dem Client API anzugeben.

Bei der JBoss EAP 6 ist dies nicht mehr möglich, da der Client-Proxy nicht mehr auf der Serverseite erstellt und nach dem Lookup dem Client übertragen wird. Der Proxy wird jetzt auf Clientseite erstellt. Diese Optimierung vermeidet einen Serveraufruf für Lookup und Klassen-Uploads.

### Entity Bean Pool Konfiguration

Entity Bean Pool Konfiguration wird in JBoss EAP 6 nicht empfohlen. Da es auf Konfiguration des **<strict-max-pool>** Elements limitiert ist, können Deadlocks und andere Probleme auftreten, wenn der Pool zu klein ist um alle Entitäts im Ergebnissatz zu laden. Entity Beans haben keine großen Lebenszyklusmethoden während der Initialisierung, daher ist die Erstellung der Instanz und umgebenden Container nicht langsamer, als wenn eine in einem Pool zusammengefasste Entity Bean Instanz benutzt wird.

### Die jboss.xml Deployment Deskriptor Datei ersetzen

Der **jboss-ejb3.xml** Deployment-Deskriptor ersetzt die **jboss.xml** Deployment-Deskriptor-Datei. Diese Datei wird bei der Außerkraftsetzung und Hinzufügung von Features des von der Java Enterprise Edition (EE) definierten **ejb-jar.xml** Deployment-Deskriptors verwendet. Die neue Datei ist nicht kompatibel mit **jboss.xml** und die **jboss.xml** wird jetzt bei Deployments ignoriert.

Wenn Sie zum Beispiel in einer früheren Release der JBoss EAP eine **<resource-ref>** in der **ejb-jar.xml** Datei definiert haben, brauchten Sie eine entsprechende Ressourcendefinition für den JNDI Namen in der **jboss.xml** Datei. XDoclet erstellte automatisch diese beiden Deployment-Deskriptor-Dateien. In JBoss EAP 6 ist die JNDI Mapping Information jetzt in der **jboss-ejb3.xml** Datei definiert. Es ist vorauszusetzen, dass die Datenquelle im Java Source Code folgendermaßen definiert ist.

```
DataSource ds1 = (DataSource) new
InitialContext().lookup("java:comp/env/jdbc/Resource1");
DataSource ds2 = (DataSource) new
InitialContext().lookup("java:comp/env/jdbc/Resource2");
```

Das **ejb-jar.xml** definiert folgende Ressourcen-Referenzen.

```
<resource-ref >
  <res-ref-name>jdbc/Resource1</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Container</res-auth>
</resource-ref>
<resource-ref >
  <res-ref-name>java:comp/env/jdbc/Resource2</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Container</res-auth>
</resource-ref>
```

Die **jboss-ejb3.xml** Datei ordnet die JNDI Namen den Referenzen zu, indem sie folgende XML Syntax benutzt.

```
<resource-ref>
  <res-ref-name>jdbc/Resource1</res-ref-name>
  <jndi-name>java:jboss/datasources/ExampleDS</jndi-name>
</resource-ref>
<resource-ref>
  <res-ref-name>java:comp/env/jdbc/Resource2</res-ref-name>
  <jndi-name>java:jboss/datasources/ExampleDS</jndi-name>
</resource-ref>
```

Manche der Konfigurationsoptionen, die in der JBoss EAP 5.x **jboss.xml** Datei verfügbar waren, wurden nicht in die JBoss EAP 6 implementiert. Die folgende Liste beschreibt einige der in der **jboss.xml** Datei allgemein verwendeten Attribute und ob es eine Alternative in der JBoss EAP 6 gibt.

- Das **method-attribute** Element wurde verwendet um individuelle Entity und Session Beans Methoden zu konfigurieren.
  - Die **read-only** und **idempotent** Konfigurationsoptionen wurden nicht in die JBoss EAP 6 portiert.
  - Die **transaction-timeout** Option wird jetzt in der **jboss-ejb3.xml** Datei konfiguriert.
- Das **missing-method-permission-exclude-mode** Attribut änderte das Verhalten der Methoden ohne die Implementierung von expliziten Sicherheits-Metadaten auf einer gesicherten Bean. In JBoss EAP 6 wird das Fehlen einer **@RolesAllowed** Annotation derzeit ähnlich gehandhabt, wie **@PermitAll**

### Datenquellen-Typ Mapping Konfiguration

In früheren Versionen der JBoss EAP war es möglich Datenquellen Typ-Mapping innerhalb der **\*-ds.xml** Datenquellen Deployment Konfigurationsdatei zu konfigurieren.

In JBoss EAP 6 muss dies in der **jbosscomp-jdbc.xml** Deployment-Deskriptor-Datei getan werden.

```
<defaults>
  <datasource-mapping>mySQL</datasource-mapping>
  <create-table>true</create-table>
  ....
</defaults>
```

In früheren Versionen der JBoss EAP wurde benutzerdefiniertes Mapping in der **standardjbosscmp-jdbc.xml** Datei vorgenommen. Diese Datei ist nicht mehr verfügbar und das Mapping wird jetzt in der **jbosscmp-jdbc.xml** Deployment-Deskriptor-Datei vorgenommen.

## Zusätzliche Änderungen der Container-Managed Persistence (CMP) und Container-Managed Relationship (CMR)

### Container Managed Relationship (CMR) Iterator und Collection Änderungen

In früheren Releases der JBoss EAP war es für manche Container, zum Beispiel für den **cmp2.x jdbc2 pm** Container, möglich CMR Collections zu durchlaufen und Relationen zu entfernen oder hinzuzufügen. Da Containerkonfiguration nicht unterstützt wird, ist das nicht mehr möglich in JBoss EAP 6. Informationen darüber, wie Sie die gleiche Funktionalität im Applikationscode erreichen, finden Sie unter [EJB2.1 Finder for CMP entities with relations \(CMR\) returns duplicates in EAP6](#) im Support Knowledgebase Solutions Abschnitt des Kundenportals .

### Container Managed Relationship (CMR) Doppelte Einträge für Finder

In früheren Versionen der JBoss EAP war es möglich verschiedene CMP Container auszuwählen, die verschiedene Persistenzstrategien benutzen. Der **cmp2.x jdbc2 pm** Container in JBoss EAP 5.x benutzte optimierte **SQL-92** um optimierte LEFT OUTER JOIN Syntax für Finder zu erzeugen. Da JBoss EAP 6.x nur die Standardcontainer für CMP und CMR unterstützt, enthält die Implementierung diese Optimierungen nicht. Der Finder sollte das Schlüsselwort **DISTINCT** im **SELECT** Statement enthalten um das kartesische Produkt im Ergebnissatz zu vermeiden. Weitere Informationen finden Sie unter [EJB2.1 Finder for CMP entities with relations \(CMR\) returns duplicates in EAP6](#) im Support Knowledgebase Solutions Abschnitt im Kundenportal.

### Löschweitergabe-Standard Änderung für CMP Entity Beans

Der Standardwert der Löschweitergabe wurde auf **false** geändert. Das kann zu Löschfehlern in JBoss EAP 6 führen. Wenn Entity Relationen als **cascade-delete** markiert sind, müssen Sie **batch-cascade-delete** explizit auf **true** in der **jbosscmp-jdbc.xml** Datei setzen. Weitere Informationen finden Sie unter [cascade delete fail for EJB2 CMP Entities after migration to EAP6](#) im Support Knowledgebase Solutions Abschnitt im Kundenportal.

### CMP benutzerdefinierte Mapper für benutzerdefinierte Felder

Wenn Sie benutzerdefinierte Mapperklassen, wie **JDBCParameterSetter**, **JDBCResultSetReader** und **Mapper** in Ihrer JBoss EAP 5.x Applikation verwendet haben, wird Ihnen vielleicht **java.lang.ClassNotFoundException** angezeigt, wenn Sie Ihre Applikation in JBoss EAP 6 deployen. Das liegt daran, dass die Paketnamen für die Interfaces von **org.jboss.ejb.plugins.cmp.jdbc.Mapper** zu **org.jboss.as.cmp.jdbc.Mapper** geändert wurden. Weitere Informationen finden Sie unter [How to use Field mapping for custom classes in an EJB2 CMP application in EAP6](#) im Support Knowledgebase Solutions Abschnitt im Kundenportal.

### Erstellung von Primärschlüsseln unter Verwendung von Entity-Befehlen

Wenn Ihre JBoss EAP 5 Applikation **entity-commands** benutzt um Primärschlüssel zu erzeugen, zum Beispiel **Sequence** oder **Auto-increment**, wird Ihnen vielleicht eine **ClassNotFoundException** für die **JDBCOracleSequenceCreateCommand** Klasse angezeigt,

wenn Sie Ihre Applikation zu JBoss EAP 6 migrieren. Das liegt daran, dass das Klassenpaket von **org.jboss.ejb.plugins.cmp.jdbc** zu **org.jboss.as.cmp.jdbc.keygen** geändert wurde. Wenn Sie diese Klasse in Ihrer JBoss EAP 6 Applikation sehen, müssen Sie auch eine Abhängigkeit auf dem **EAP\_HOME/modules/system/layers/base/org/jboss/as/cmp** Modul hinzufügen.

## Applikations-Änderungen

### Bearbeiten Sie den Code zur Verwendung der neuen JNDI Namespace-Regeln.

Wie beim EJB 3.0 müssen Sie den vollständigen JNDI-Präfix mit EJB 2.x verwenden. Weitere Informationen zu den neuen JNDI-Namespace-Regeln und Code-Beispiele finden Sie unter [Abschnitt 3.1.8.1, »Aktualisierung der JNDI Namespace-Namen der Applikation«](#).

Beispiele, die zeigen, wie JNDI-Namespace früherer Releases aktualisiert werden, finden Sie hier: [Abschnitt 3.1.8.5, »Beispiele von JNDI-Namespace in früheren Releases und wie diese in der JBoss EAP 6 spezifiziert sind«](#).

### Bearbeiten Sie den `jboss-web.xml`-Dateideskriptor

Bearbeiten Sie `<jndi-name>` für jedes `<ejb-ref>` zur Verwendung des neuen JNDI Lookup-Formats.

### Benutzen Sie XDoclet um den JNDI Name von Internen Lokalen Interfaces zuzuordnen

Mit EJB 2 war es üblich über das **Locator** Muster Beans zu suchen. Wenn Sie dieses Muster in Ihrer Applikation benutzt haben, anstatt den Applikationscode zu ändern, können Sie mit [XDoclet](#) eine Map für die neuen JNDI-Namen erstellen.

Eine typische XDoclet Annotation sieht folgendermaßen aus:

```
@ejb.bean name="UserAttribute" display-name="UserAttribute" local-jndi-name="ejb21/UserAttributeEntity" view-type="local" type="CMP" cmp-version="2.x" primkey-field="id"
```

Der JNDI Name **ejb21/UserAttributeEntity** im angeführten Beispiel ist in JBoss EAP 6 nicht mehr gültig. Sie können diesen Namen unter Verwendung des **naming** Subsystems in der Serverkonfiguration und einem Patch für XDoclet einem gültigen JNDI-Namen zuordnen.

Sie können benutzerdefinierte Mapper erstellen, wie es im vorherigen Paragraph *CMP Customized Mappers for Custom Fields* beschrieben wurde, oder Sie können den Code entsprechend folgender Vorgehensweise ändern.

### Prozedur 3.24. Den XDoclet Generierten Code ändern und das Benennungs-Subsystem verwenden

1. Extrahieren Sie die XDoclet **lookup.xdt** Vorlage im **ejb-module.jar** und modifizieren Sie das **lookup()** im **lookupHome** wie folgt:

```
private static Object lookupHome(java.util.Hashtable environment,
String jndiName, Class narrowTo) throws
javax.naming.NamingException {
    // Obtain initial context
    javax.naming.InitialContext initialContext = new
javax.naming.InitialContext(environment);
    try {
        // Replace the existing lookup
```

```

        // Object objRef = initialContext.lookup(jndiName);
        // This is the new mapped lookup
        Object objRef;
        try {
            // try JBoss EAP mapping
            objRef = initialContext.lookup("global/"+jndiName);
        } catch (java.lang.Exception e) {
            objRef = initialContext.lookup(jndiName);
        }
        // only narrow if necessary
        if (java.rmi.Remote.class.isAssignableFrom(narrowTo))
            return javax.rmi.PortableRemoteObject.narrow(objRef,
narrowTo);
        else
            return objRef;
    } finally {
        initialContext.close();
    }
}

```

2. Führen Sie Ant aus, während Sie das Vorlagenattribut auf Benutzung des modifizierten **lookup.xdt** für die **ejbdoclet** Aufgabe einstellen.
3. Modifizieren Sie das **naming** Subsystem in der Server-Konfigurationsdatei um den alten JNDI-Namen dem neuen, gültigen JNDI-Namen zuzuordnen.

```

<subsystem xmlns="urn:jboss:domain:naming:1.2">
    <bindings>
        <lookup name="java:global/ejb21/UserAttributeEntity"
lookup="java:global/ejb2CMP/ejb/UserAttribute!de.wfink.ejb21.cmp.c
mr.UserAttributeLocalHome"/>
    </bindings>
    <remote-naming/>
</subsystem>

```

## Zusammenfassung von veralteten Dateien

Folgende Dateien werden in der JBoss EAP 6 nicht mehr unterstützt.

### **jboss.xml**

Die **jboss.xml** Deployment-Deskriptor-Datei wird nicht mehr unterstützt und wird ignoriert, wenn sie im deployten Archiv erscheint.

### **standardjbosscmp-jdbc.xml**

Die **standardjbosscmp-jdbc.xml** Konfigurationsdatei wird nicht mehr unterstützt. Diese Konfigurationsinformation ist jetzt im **org.jboss.as.cmp** Modul erfasst und ist nicht mehr anpassbar.

### **standardjboss.xml**

Die **standardjboss.xml** Konfigurationsdatei wird nicht mehr unterstützt. Diese Konfigurationsinformation ist jetzt in der **standalone.xml** Datei erfasst, wenn ein Standalone Server ausgeführt wird, oder in der **domain.xml** Datei, wenn sie in einer Managed Domain ausgeführt wird.

[Fehler melden](#)

## 3.2.12. JBoss AOP Änderungen

### 3.2.12.1. Aktualisierung von Applikationen, die JBoss AOP verwenden

JBoss AOP (Aspect Oriented Programming) ist in der JBoss EAP 6 nicht mehr enthalten. In früheren Releases wurde JBoss AOP vom EJB-Container benutzt. In der JBoss EAP 6 verwendet der EJB-Container einen neuen Mechanismus. Falls Ihre Applikation JBoss AOP verwendet, so müssen Sie Ihren Applikationscode wie folgt bearbeiten.

#### Refakturieren der Applikation

- Standard EJB3-Konfigurationen, die bislang in der **ejb3-interceptors-aop.xml**-Datei gemacht wurden, erfolgen nun in der Serverkonfigurationsdatei. Für einen Standalone Server ist dies die **standalone/configuration/standalone-full.xml**-Datei. Falls Ihr Server in einer Managed Domain läuft, so ist es die **domain/configuration/domain.xml**-Datei.
- Serverseitige AOP Interzeptoren sollten unter Verwendung der Standard Java EE **Interceptor** geändert werden. Weitere Informationen über Container Interzeptoren und darüber, wie man einen clientenseitigen Interzeptor in einer Applikation benutzt, finden Sie im Kapitel *Container Interceptors* im *Development Guide* für JBoss EAP 6 im Kundenportal unter [https://access.redhat.com/site/documentation/JBoss\\_Enterprise\\_Application\\_Platform/](https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/).

#### Verwenden der JBoss AOP Bibliotheken

- Wenn Sie den Code nicht refakturieren können, können Sie eine Kopie der JBoss AOP Bibliotheken beziehen und sie mit der Applikation zusammenstellen. Die AOP Bibliotheken funktionieren vielleicht in JBoss EAP 6, aber sie sind nicht deployt. Sie können sie manuell deployen, indem Sie folgendes Befehlszeilenargument beim Start ihres Servers eingeben: -  
**Djboss.aop.path=PATH\_TO\_AOP\_CONFIG**



#### ANMERKUNG

Auch wenn die JBoss AOP Bibliotheken in JBoss EAP 6 funktionieren, ist dies keine unterstützte Konfiguration.

[Fehler melden](#)

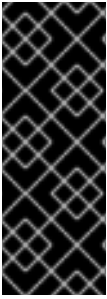
## 3.2.13. Migration von Seam 2.2 Applikationen

### 3.2.13.1. Migration der Seam 2.2 Archive zur JBoss EAP 6

#### Überblick

Wenn Sie eine Seam 2.2 Applikation migrieren, so müssen Sie die Datenquelle konfigurieren und mögliche Modulabhängigkeiten festlegen. Sie müssen außerdem festlegen, ob die Applikation Abhängigkeiten von Archiven besitzt, die nicht mit der JBoss EAP 6 geliefert werden und etwaige abhängige JARs in das **lib/**-Verzeichnis der Applikation kopieren.





## WICHTIG

Applikationen, die Hibernate direkt mit Seam 2.2 verwenden, können eine innerhalb der Applikation gepackte Version von Hibernate 3 verwenden. Hibernate 4, welches mittels des org.hibernate Moduls der JBoss EAP 6 bereitgestellt wird, wird von Seam 2.2 nicht unterstützt. Dieses Beispiel soll Ihnen dabei helfen, Ihre Applikation auf der JBoss EAP 6 in Betrieb zu nehmen. Bitte beachten Sie, dass das Packen von Hibernate 3 mit einer Seam 2.2 Applikation keine unterstützte Konfiguration ist.

### Prozedur 3.25. Migration der Seam 2.2 Archive

#### 1. Aktualisierung der Datenquellen-Konfiguration

Einige Seam 2.2 Beispiele verwenden die standardmäßige JDBC-Datenquelle namens **java:/ExampleDS**. Diese Standard-Datenquelle hat sich bei der JBoss EAP 6 zu **java:jboss/datasources/ExampleDS** geändert. Falls Ihre Applikation die Beispieldatenbank verwendet, so so können Sie eine der folgenden Vorgehensweisen wählen:

- o Falls Sie die mit der JBoss EAP 6 gelieferte Beispieldatenbank verwenden möchten, so bearbeiten Sie die **META-INF/persistence.xml**-Datei, damit sie das bestehende **jta-data-source**-Element mit dem Datenquellen JNDI-Namen der Beispieldatenbank ersetzt:

```
<!-- <jta-data-source>java:/ExampleDS</jta-data-source> -->
<jta-data-source>java:jboss/datasources/ExampleDS</jta-data-
source>
```

- o Falls Sie Ihre bestehende Datenbank lieber behalten möchten, so können Sie die Datenquellendefinition der **EAP\_HOME/standalone/configuration/standalone.xml**-Datei hinzufügen.



## WICHTIG

Sie müssen den Server stoppen, ehe Sie die Server Konfigurations-Datei bearbeiten, damit Ihre Änderungen bei einem Server-Neustart als dauerhaft erstellt werden.

Die folgende Definition ist eine Kopie der standardmäßigen HSQL Datenquelle, die in der JBoss EAP 6 definiert ist:

```
<datasource name="ExampleDS" jndi-name="java:/ExampleDS"
enabled="true" jta="true" use-java-context="true" use-ccm="true">
  <connection-url>jdbc:h2:mem:test;DB_CLOSE_DELAY=-
1</connection-url>
  <driver>h2</driver>
  <security>
    <user-name>sa</user-name>
    <password>sa</password>
  </security>
</datasource>
```

- o Sie können die Datenquellen-Definition auch mittels der Management-CLI-Befehlszeilen-Interface hinzufügen. Nachfolgend sehen Sie ein Beispiel für die Syntax, mit der Sie eine Datenquelle hinzufügen. Der "\n" am Ende der Zeile zeigt die Fortsetzung des Befehls in der folgenden Zeile an.



**Beispiel 3.5. Beispiel der Syntax für die Hinzufügung der Datenquellen-Definition**

```
$ EAP_HOME/bin/jboss-cli --connect
[standalone@localhost:9999 /] data-source add --name=ExampleDS
--jndi-name=java:/ExampleDS \
--connection-url=jdbc:h2:mem:test;DB_CLOSE_DELAY=-1 --
driver-name=h2 \
--user-name=sa --password=sa
```

Weitere Informationen zur Konfiguration einer Datenquelle finden Sie unter [Abschnitt 3.1.6.2, »Aktualisierung der Datenquellen-Konfiguration«](#).

**2. Fügen Sie die erforderlichen Abhängigkeiten hinzu**

Da Seam 2.2 Applikationen JSF 1.2 verwenden, müssen Sie Abhängigkeiten für die JSF 1.2 Module hinzufügen und die JSF 2.0 Module ausschließen. Um dies zu bewerkstelligen, müssen Sie eine **jboss-deployment-structure.xml**-Datei im **META-INF**-Verzeichnis des EAR erstellen, das folgende Daten enthält:

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-
structure:1.0">
  <deployment>
    <dependencies>
      <module name="javax.faces.api" slot="1.2" export="true"/>
      <module name="com.sun.jsf-impl" slot="1.2"
export="true"/>
    </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
      <module name="javax.faces.api" slot="main"/>
      <module name="com.sun.jsf-impl" slot="main"/>
    </exclusions>
    <dependencies>
      <module name="javax.faces.api" slot="1.2"/>
      <module name="com.sun.jsf-impl" slot="1.2"/>
    </dependencies>
  </sub-deployment>
</jboss-deployment-structure>
```

Falls Ihre Applikation Protokollierungs-Frameworks von Drittanbietern verwendet, so müssen Sie diese Abhängigkeiten wie hier beschrieben hinzufügen: [Abschnitt 3.1.4.1, »Bearbeitung von Protokollierungsabhängigkeiten«](#).

**3. Falls Ihre Applikation Hibernate 3.x verwendet, so versuchen Sie die Applikation zunächst unter Verwendung von Hibernate 4 Bibliotheken auszuführen**

Falls Ihre Applikation Seam Managed Persistence Context, Hibernate Suche, Validierung und andere Features, die sich in Hibernate 4 geändert haben, nicht verwendet, so können Sie möglicherweise die Hibernate 4 Bibliotheken verwenden. Falls Sie jedoch

**ClassNotFoundExceptions** oder **ClassCastExceptions** sehen, die auf Hibernate Klassen verweisen oder Fehler ähnlich dem folgenden sehen, so folgen Sie den Anweisungen im nachfolgenden Schritt und bearbeiten Sie die Applikation dahingehend, dass sie Hibernate 3.3 Bibliotheken verwendet.

```
Caused by: java.lang.LinkageError: loader constraint
```

violation in interface itable initialization: when resolving method "org.jboss.seam.persistence.HibernateSessionProxy.getSession(Lorg/hibernate/EntityMode;)Lorg/hibernate/Session;" the class loader (instance of org/jboss/modules/ModuleClassLoader) of the current class, org/jboss/seam/persistence/HibernateSessionProxy, and the class loader (instance of org/jboss/modules/ModuleClassLoader) for interface org/hibernate/Session have different Class objects for the type org/hibernate/Session used in the signature

#### 4. Kopieren Sie abhängige Archive von außerhalb liegenden Frameworks oder anderen Orten

Falls Ihre Applikation Hibernate 3.x verwendet und Sie Hibernate 4 nicht erfolgreich mit Ihrer Applikation verwenden können, so müssen Sie die Hibernate 3.x JARs in das **/lib**-Verzeichnis kopieren und das Hibernate-Modul im Deployments-Abschnitt der **META-INF/jboss-deployment-structure.xml** wie folgt ausschließen:

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.0">
  <deployment>
    <exclusions>
      <module name="org.hibernate"/>
    </exclusions>
  </deployment>
</jboss-deployment-structure>
```

Hier sind zusätzliche Schritte, die Sie beim bündeln von Hibernate 3.x mit Ihrer Applikation vornehmen müssen. Weitere Informationen finden Sie unter [Abschnitt 3.2.2.2, »Konfiguration der Änderungen bei Applikationen, die Hibernate und JPA verwenden«](#).

#### 5. Fehlerbehebung und Auflösung von Seam 2.2 JNDI-Fehlern

Bei der Migration einer Seam 2.2 Applikation kann es sein, dass Sie **javax.naming.NameNotFoundException**-Fehler wie den folgenden im Protokoll sehen:

```
javax.naming.NameNotFoundException: Name 'jboss-seam-booking' not
found in context ''
```

Falls Sie JNDI-Lookups nicht durch den Code hindurch bearbeiten wollen, so können Sie die **components.xml**-Datei der Applikation wie folgt bearbeiten:

##### a. Ersetzen Sie das bestehende core-init Element

Zuerst müssen Sie das bestehende core-init Element wie folgt ersetzen:

```
<!-- <core:init jndi-pattern="jboss-seam-booking/#
{ejbName}/local" debug="true" distributable="false"/> -->
<core:init debug="true" distributable="false"/>
```

##### b. Finden Sie die JNDI-Binding INFO-Nachrichten im Serverprotokoll

Anschließend finden Sie die JNDI-Binding INFO-Nachrichten im Serverprotokoll, die beim Deployment der Applikation gedruckt wurden. Die JNDI-Binding-Nachrichten sollten in etwa so aussehen:

```
INFO
org.jboss.as.ejb3.deployment.processors.EjbJndiBindingsDeployment
UnitProcessor (MSC service thread 1-1) JNDI bindings for session
```

```

bean
named AuthenticatorAction in deployment unit subdeployment
"jboss-seam-booking.jar" of deployment "jboss-seam-booking.ear"
are as follows:
    java:global/jboss-seam-booking/jboss-seam-
booking.jar/AuthenticatorAction!org.jboss.seam.example.booking.Au
thenticator
    java:app/jboss-seam-
booking.jar/AuthenticatorAction!org.jboss.seam.example.booking.Au
thenticator

java:module/AuthenticatorAction!org.jboss.seam.example.booking.Au
thenticator
    java:global/jboss-seam-booking/jboss-seam-
booking.jar/AuthenticatorAction
    java:app/jboss-seam-booking.jar/AuthenticatorAction
    java:module/AuthenticatorAction

```

### c. Fügen Sie Komponentenelemente hinzu

Für jede JNDI-Binding INFO-Nachricht im Protokoll fügen Sie dem **components.xml**-Datei ein entsprechendes **component**-Element hinzu:

```

<component
class="org.jboss.seam.example.booking.AuthenticatorAction" jndi-
name="java:app/jboss-seam-booking.jar/AuthenticatorAction" />

```

Weitere Informationen zur Fehlerbehebung und Auflösung von Migrationsproblemen finden Sie unter [Abschnitt 4.2.1, »Fehler- und Problembehebung bei der Migration«](#).

Eine Liste bekannter Migrationsprobleme mit Seam 2 Archiven finden Sie unter [Abschnitt 3.2.13.2, »Seam 2.2 Archiv-Migrationsprobleme«](#).

## Ergebnis

Das Seam 2.2 Archiv deployt und läuft nun auf der JBoss EAP 6.

[Fehler melden](#)

### 3.2.13.2. Seam 2.2 Archiv-Migrationsprobleme

#### Seam 2.2 Drools und Java 7 sind nicht kompatibel

Seam 2.2 Drools und Java 7 sind inkompatibel und schlagen fehl mit der Ausnahme `org.drools.RuntimeDroolsException: value '1.7' is not a valid language level error`.

#### Seam 2.2.5 signiertes **cglib.jar** hindert das Spring Beispiel am Laufen

Wenn das Spring Beispiel mittels des signierten **cglib.jar** betrieben wird, das mit der Seam 2.2.5 in der JBoss EAP 5 geliefert wird, so schlägt es mit der folgenden Grundursache fehl:

```

java.lang.SecurityException: class
"org.jboss.seam.example.spring.UserService$$EnhancerByCGLIB$$7d6c3d12"'s
signer information does not match signer information of other classes in
the same package

```

Sie umgehen dieses Problem, indem Sie die **cglib.jar** wie folgt unsignieren:

```
zip -d $SEAM_DIR/lib/cglib.jar META-INF/JBOSSCOD\*
```

### Seambay Beispiel schlägt fehl mit `NotLoggedInException`

Die Ursache für dieses Problem liegt darin, dass die Kopfzeile der SOAP-Nachricht Null ist, wenn die Nachricht im `SOAPRequestHandler` verarbeitet wird und daher die Conversation-ID nicht eingestellt ist.

Sie umgehen dieses Problem, indem Sie die

**`org.jboss.seam.webservice.SOAPRequestHandler.handleOutbound`** wie in

<https://issues.jboss.org/browse/JBPAPP-8376> beschrieben außer Kraft setzen.

### Seambay Beispiel schlägt fehl mit `UnsupportedOperationException: no transaction`

Dieser Fehler wird durch Änderungen im JNDI-Namen der `UserTransaction` in der JBoss EAP 6 verursacht.

Sie umgehen dieses Problem, indem Sie die

**`org.jboss.seam.transaction.Transaction.getUserTransaction`** wie in

<https://issues.jboss.org/browse/JBPAPP-8322> beschrieben außer Kraft setzen.

### Tasks-Beispiel meldet `org.jboss.resteasy.spi.UnhandledException: Unable to unmarshall request body`

Dieser Fehler wurzelt in der Inkompatibilität von `seam-resteasy-2.2.5` (das in der JBoss EAP 5.1.2 enthalten ist) und dem `RESTEasy 2.3.1.GA` der JBoss EAP 6.

Sie umgehen dieses Problem, indem Sie **`jboss-deployment-structure.xml`** zum Ausschluss von `resteasy-jaxrs`, `resteasy-jettison-provider` und `resteasy-jaxb-provider` aus dem Haupt-Deployment und `resteasy-jaxrs`, `resteasy-jettison-provider`, `resteasy-jaxb-provider` und `resteasy-yaml-provider` aus dem **`jboss-seam-tasks.war`** verwenden, wie in <https://issues.jboss.org/browse/JBPAPP-8315> beschrieben. Es ist dann notwendig die `RESTEasy`-Bibliotheken mit einzuschließen, die mit Seam 2.2 im EAR gebündelt sind.

### Deadlock-Situation zwischen `org.jboss.seam.core.SynchronizationInterceptor` und `stateful component instance EJB lock during an AJAX request`

Eine Fehlerseite mit "Caused by `javax.servlet.ServletException` with message: "javax.el.ELException: /main.xhtml @36,71 value="#{hotelSearch.pageSize}": `org.jboss.seam.core.LockTimeoutException`: could not acquire lock on @Synchronized component: `hotelSearch`" oder eine ähnliche Fehlermeldung wird angezeigt.

Das Problem ist, dass Seam 2 seine eigenen Sperrvorgänge außerhalb der Stateful Session Bean (SFSB) Sperre und mit einem anderen Bereich durchführt. Das bedeutet, dass wenn ein anderer Thread innerhalb derselben Transaktion zweimal auf ein EJB zugreift, er nach dem ersten Aufruf eine SFSB-Sperre, nicht aber die Seam-Sperre hat. Ein zweiter Thread kann dann die Seam-Sperre erhalten, die dann auf die EJB-Sperre trifft und wartet. Wenn der erste Thread den zweiten Aufruf versucht, so blockiert er am Seam 2 Interzeptor und es tritt eine Deadlock-Situation auf. In Java EE 5 würden EJBs bei gleichzeitigem Zugriff sofort eine Ausnahme melden. Dieses Verhalten hat sich bei Java EE 6 geändert.

Sie umgehen dieses Problem, indem Sie dem EJB `@AccessTimeout(0)` hinzufügen. Dies führt zur sofortigen Meldung einer **`ConcurrentAccessException`** wenn diese Situation auftritt.

### Dvdstore-Beispiel `create-Ordner` schlägt mit `javax.ejb.EJBTransactionRolledbackException` fehl

Dvdstore-Beispiel `create-Ordner` schlägt mit dem folgenden Fehler fehl:

JBAS011437: Found extended persistence context in SFSB invocation call stack but that cannot be used because the transaction already has a transactional context associated with it. This can be avoided by changing application code, either eliminate the extended persistence context or the transactional context. See JPA spec 2.0 section 7.6.3.1.

Dieses Problem tritt aufgrund von Änderungen in der JPA-Spezifikation auf.

Die Fehlerbehebung hierfür besteht in der Änderung des Persistenzkontexts zu **transactional** in den **CheckoutAction**- und **ShowOrdersAction**-Klassen und der Verwendung des Entity-Manager Vereinigungsvorgangs in den **cancelOrder**- und **detailOrder**-Methoden.

### Der JBoss Cache Seam Cache Provider kann nicht bei der JBoss EAP 6 verwendet werden

JBoss Cache wird von der JBoss EAP 6 nicht unterstützt. Dies führt dazu, dass der JBoss Cache Seam Cache Provider in einer Seam Applikation am Applikationsserver fehlschlägt.

```
java.lang.NoClassDefFoundError: org/jboss/util/xml/JBossEntityResolver
```

### Hibernate 3.3.x Auto-Scan für JPA-Entities Probleme mit der JBoss EAP 6

Sie beheben diesen Fehler, indem Sie alle Entity-Klassen manuell in der persistence.xml-Datei auflisten. Zum Beispiel:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
version="1.0">
  <persistence-unit name="example_pu">
    <description>Hibernate 3 Persistence Unit.</description>
    <jta-data-source>java:jboss/datasources/ExampleDS</jta-data-
source>
    <properties>
      <property name="jboss.as.jpa.providerModule"
value="hibernate3-bundled" />
    </properties>
    <class>com.acme.Foo</class>
    <class>com.acme.Bar</class>
  </persistence-unit>
</persistence>
```

### Das Abrufen von EJB Seam Komponenten von non-EJB Threads führt zu einer javax.naming.NameNotFoundException

Dieses Problem ist die Folge von Änderungen in der JBoss EAP 6 bezüglich der Implementierung eines neuen modularen Klassenladesystems und bezüglich der Anwendung der neuen, standardisierten JNDI-Namespace Konventionen. Der **java:app** Namespace ist festgelegt für Namen, die von allen Komponenten in einer einzigen Applikation geteilt werden. Non-EE Threads, wie Quartz asynchrone Threads, müssen den **java:global** Namespace benutzen, der von allen Applikationen geteilt wird, die in einer Applikations-Server Instanz deployt sind.

Wenn Sie beim Versuch EJB Seam Komponenten von Quartz asynchronen Methoden abzurufen eine **javax.naming.NameNotFoundException** erhalten, so müssen Sie die **components.xml** Datei dahingehend modifizieren, den globalen JNDI-Namen zu benutzen, zum Beispiel:

■

```
<component class="org.jboss.seam.example.quartz.MyBean" jndi-  
name="java:global/seam-quartz/quartz-ejb/myBean"/>
```

Weitere Informationen zu JNDI-Änderungen finden Sie unter folgendem Oberbegriff: [Abschnitt 3.1.8.1, »Aktualisierung der JNDI Namespace-Namen der Applikation«](#). Weitere Informationen zu diesem speziellen Problem finden Sie unter *BZ#948215 - Seam2.3 javax.naming.NameNotFoundException trying to call EJB Seam components from quartz asynchronous methods* in den *2.2.0 Release Notes* für *Red Hat JBoss Web Framework Kit* im Red Hat Kundenportal.

[Fehler melden](#)

## 3.2.14. Migration von Spring Applikationen

### 3.2.14.1. Migration von Spring Applikationen

Informationen zur Migration von Spring Applikationen finden Sie in der *Red Hat JBoss Web Framework Kit* Dokumentation im Kundenportal unter <https://access.redhat.com/site/documentation/>. Suchen Sie nach **Red Hat JBoss Middleware**, klicken Sie dann auf den *Red Hat JBoss Web Framework Kit* Link. Der *Spring Installation Guide* und *Spring Developer Guide* sind in mehreren Formaten verfügbar.

[Fehler melden](#)

## 3.2.15. Andere Änderungen, die Ihre Migration beeinflussen können

### 3.2.15.1. Machen Sie sich mit anderen Änderungen vertraut, die Einfluss auf Ihre Migration haben können

Nachfolgend finden Sie eine Liste von Änderungen in der JBoss EAP 6, die Ihre Migration beeinflussen können.

- [Abschnitt 3.2.15.2, »Änderung des Maven Plug-in Namens«](#)
- [Abschnitt 3.2.15.3, »Bearbeitung von Client-Applikationen«](#)

[Fehler melden](#)

### 3.2.15.2. Änderung des Maven Plug-in Namens

Das **jboss-maven-plugin** wurde nicht aktualisiert und funktioniert nicht bei der JBoss EAP 6. Sie müssen jetzt **org.jboss.as.plugins:jboss-as-maven-plugin** für das Deployment in das richtige Verzeichnis verwenden.

[Fehler melden](#)

### 3.2.15.3. Bearbeitung von Client-Applikationen

Falls Sie die Migration einer Client-Applikationen planen, die sich mit der JBoss EAP 6 verbinden, so sollten Sie sich dessen bewusst sein, dass sich der Name und Speicherort des JAR, das die Client-Bibliotheken bündelt geändert hat. Dieses JAR heißt jetzt **jboss-client.jar** und befindet sich im **EAP\_HOME/bin/client/-**Verzeichnis. Es ersetzt **EAP\_HOME/client/jbossall-client.jar** und enthält alle erforderlichen Abhängigkeiten für die Verbindung zur JBoss EAP 6 von einem Remote Client.

[Fehler melden](#)

## KAPITEL 4. TOOLS UND TIPPS

### 4.1. RESSOURCEN, DIE BEI DER MIGRATION HELFEN SOLLEN

#### 4.1.1. Ressourcen, die bei Ihrer Migration helfen sollen

Folgendes ist eine Liste von Ressourcen, die Ihnen bei der Migration von Applikationen auf die JBoss EAP 6 helfen sollen.

##### Tools

Es gibt mehrere Tools, die bei der Automatisierung einiger der Konfigurationsänderungen helfen. Weitere Informationen finden Sie unter: [Abschnitt 4.1.2, »Machen Sie sich mit den Tools vertraut, die Ihnen bei der Migration helfen können«](#).

##### Tipps zur Fehlerbehebung

Eine Liste der häufigsten Ursachen und der Behebungen von Problemen und Fehlern, die bei Ihrer Migration Ihrer Applikation auftreten können, finden Sie hier: [Abschnitt 4.2.1, »Fehler- und Problembehebung bei der Migration«](#).

##### Beispielmigrationen

Beispiele von Applikationen, die zur JBoss EAP 6 migriert wurde finden Sie hier: [Abschnitt 4.3.1, »Übersicht der Migration der Beispiellapplikationen«](#).

[Fehler melden](#)

### 4.1.2. Machen Sie sich mit den Tools vertraut, die Ihnen bei der Migration helfen können

#### Zusammenfassung

Es gibt mehrere Tools, die Ihnen bei der Migration helfen. Nachfolgend sehen Sie eine Liste dieser Tools einschließlich einer Beschreibung davon, was diese tun.

##### Tattletale

Aufgrund der Änderung am modularen Klassenladen müssen Sie Applikationsabhängigkeiten finden und korrigieren. Tattletale kann Ihnen bei der Identifizierung abhängiger Modulnamen helfen und die Konfigurations-XML für Ihre Applikation generieren.

[Abschnitt 4.1.3, »Verwendung von Tattletale zum Auffinden von Applikationsabhängigkeiten«](#)

##### IronJacamar Migrationstool

Bei der JBoss EAP 6 werden Datenquellen und Ressourcenadapter nicht mehr in einer separaten Datei konfiguriert. Sie werden jetzt in der Serverkonfigurationsdatei konfiguriert und verwenden neue Schemas. Das IronJacamar Migrationstool kann Ihnen bei der Konvertierung der alten Konfiguration in das von der JBoss EAP 6 erwartete Format helfen.

[Abschnitt 4.1.6, »Verwendung des IronJacamar Migrationstools zur Migration der Datenquellen- und Ressourcenadapterkonfigurationen«](#)

[Fehler melden](#)



### 4.1.3. Verwendung von Tattletale zum Auffinden von Applikationsabhängigkeiten

#### Zusammenfassung

Aufgrund modularer Klassenladeänderungen bei der JBoss EAP 6 können **ClassNotFoundException** oder **ClassCastException** im JBoss-Protokoll auftreten, wenn Sie Ihre Applikation migrieren. Um diese Fehler zu beheben, müssen Sie die JARs auffinden, die die von den Ausnahmen spezifizierten Klassen enthalten.

Tattletale ist ein ausgezeichnetes Drittanbieter-Tool, das Ihre Applikation rekursiv scannt und ausführliche Berichte zu deren Inhalten liefert. Tattletale 1.2.0.Beta2 oder später enthält zusätzlichen Support um mit dem neuen JBoss Modules Klassenladen in JBoss EAP 6 zu helfen. Tatttales "JBoss AS 7" Bericht kann zur automatischen Identifizierung und Generierung abhängiger Modulnamen zum Einschluss in die **jboss-deployment-structure.xml**-Datei Ihrer Applikation verwendet werden.

#### Prozedur 4.1. Installation und Ausführung von Tattletale zum Auffinden von Applikationsabhängigkeiten

1. [Abschnitt 4.1.4, »Download und Installation von Tattletale«](#)
2. [Abschnitt 4.1.5, »Erstellen und Prüfen des Tattletale Berichts«](#)



#### ANMERKUNG

Tattletale ist ein Drittanbieter Tool und wird nicht als Teil von JBoss EAP 6 unterstützt. Die aktuellste Dokumentation zur Installation und Verwendung von Tattletale finden Sie auf der Tattletale Website unter <http://tattletale.jboss.org/>.

[Fehler melden](#)

### 4.1.4. Download und Installation von Tattletale

#### Prozedur 4.2. Download und Installation von Tattletale

1. Laden Sie sich die Tattletale Version 1.2.0.Beta2 oder neuer unter <http://sourceforge.net/projects/jboss/files/JBoss%20Tattletale>.
2. Entpacken Sie die Datei im Verzeichnis Ihrer Wahl.
3. Bearbeiten Sie die **TATTLETALE\_HOME/jboss-tattletale.properties**-Datei wie folgt:
  - a. Fügen Sie der **profiles**-Property **ee6** und **as7** hinzu.

```
profiles=java5, java6, ee6, as7
```

- b. Entfernen Sie die Kommentierung aus den **scan**- und **reports**-Properties.

[Fehler melden](#)

### 4.1.5. Erstellen und Prüfen des Tattletale Berichts

1. Erstellen Sie mittels des folgenden Befehls einen Tattletale Bericht: **java -jar TATTLETALE\_HOME/tattletale.jar APPLICATION\_ARCHIVE OUTPUT\_DIRECTORY**

Zum Beispiel: `java -jar tattletale-1.2.0.Beta2/tattletale.jar  
~/applications/jboss-seam-booking.ear ~/output-results/`

2. Öffnen Sie in einem Browser die **OUTPUT\_DIRECTORY/index.html**-Datei und klicken Sie auf "JBoss AS 7" unter dem "Reports"-Abschnitt.
  - a. Die linke Spalte listet die von der Applikation verwendeten Archive. Klicken Sie auf das **ARCHIVE\_NAME**-Link um weitere Informationen zu dem Archiv zu erhalten, so etwa zum Speicherort, Manifestinformationen und Klassen, die es enthält.
  - b. Das **jboss-deployment-structure.xml**-Link in der rechten Spalte zeigt wie die Modulabhängigkeit für das in der linken Spalte genannte Archiv festgelegt wird. Klicken Sie auf dieses Link um zu sehen, wie die Informationen zum Deployment-Abhängigkeitsmodul für dieses Archiv definiert werden.

[Fehler melden](#)

#### 4.1.6. Verwendung des IronJacamar Migrationstools zur Migration der Datenquellen- und Ressourcenadapterkonfigurationen

##### Zusammenfassung

In früheren Versionen des Applikationsservers wurden Datenquellen und Ressourcenadapter unter Verwendung einer Datei mit dem Suffix **\*-ds.xml** konfiguriert und deployt. Die IronJacamar 1.1 Distribution enthält ein Migrationstool, das zur Konvertierung dieser Konfigurationsdateien in das von der JBoss EAP 6 erwartete Format verwendet werden kann. Das Tool parst die Quellkonfigurationsdatei der vorherigen Release, erstellt und schreibt dann die XML-Konfiguration in eine Output-Datei im neuen Format. Diese XML kann dann kopiert und im korrekten Untersystem in der JBoss EAP 6 Serverkonfigurationsdatei eingefügt werden. Dieses Tool versucht möglichst effektiv alte Attribute und Elemente in das neue Format zu konvertieren, jedoch ist es möglich, dass zusätzliche Änderungen an der generierten Datei vorgenommen werden müssen.

##### Prozedur 4.3. Installation und Ausführung des IronJacamar Migrationstools

1. [Abschnitt 4.1.7, »Download und Installation des IronJacamar Migrationstools«](#)
2. [Abschnitt 4.1.8, »Verwenden Sie das IronJacamar Migrationstool für die Konvertierung einer Datenquellen-Konfigurationsdatei «](#)
3. [Abschnitt 4.1.9, »Verwenden Sie das IronJacamar Migrationstool für die Konvertierung einer Ressourcenadapter-Konfigurationsdatei «](#)



##### ANMERKUNG

Das IronJacamar Migrationstool ist ein Drittanbieter-Tool und wird nicht als Teil der JBoss EAP 6 unterstützt. Weitere Informationen zu IronJacamar finden Sie unter <http://www.ironjacamar.org/>. Die aktuellste Dokumentation zur Installation und Verwendung des Tools finden Sie unter <http://www.ironjacamar.org/documentation.html>.

[Fehler melden](#)

#### 4.1.7. Download und Installation des IronJacamar Migrationstools



## ANMERKUNG

Das Migrationstool ist nur in IronJacamar 1.1 Version oder höher verfügbar und erfordert Java 7 oder höher.

1. Laden Sie die neuste Ausgabe von IronJacamar hier herunter:  
<http://www.ironjacamar.org/download.html>
2. Entpacken Sie die heruntergeladene Datei im Verzeichnis Ihrer Wahl.
3. Finden Sie das Converter-Skript in der IronJacamar\_Distribution.
  - Das Linux-Skript befindet sich hier: **`IRONJACAMAR_HOME/doc/as/converter.sh`**
  - Die Windows Batch-Datei befindet sich hier:  
**`IRONJACAMAR_HOME/doc/as/converter.bat`**

[Fehler melden](#)

### 4.1.8. Verwenden Sie das IronJacamar Migrationstool für die Konvertierung einer Datenquellen-Konfigurationsdatei



## ANMERKUNG

Das IronJacamar Converter-Skript erfordert Java 7 oder höher.

#### Prozedur 4.4. Konvertierung einer Datenquellen-Konfigurationsdatei

1. Öffnen Sie eine Befehlszeile und navigieren Sie zum **`IRONJACAMAR_HOME/doc/as/-`** Verzeichnis.
2. Führen Sie das Converter-Skript durch Eingabe des folgenden Befehls aus:
  - Für Linux: **`./converter.sh -ds SOURCE_FILE TARGET_FILE`**
  - Für Microsoft Windows: **`./converter.bat -ds SOURCE_FILE TARGET_FILE`**

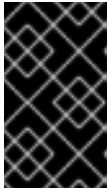
Die **`SOURCE_FILE`** ist die Datenquelle -ds.xml-Datei aus der vorherigen Release. Die **`TARGET_FILE`** enthält die neue Konfiguration.

Um zum Beispiel die sich im aktuellen Verzeichnis befindende **`jboss-seam-booking-ds.xml`**-Datenquellen-Konfigurationsdatei zu konvertieren, würden Sie folgendes eingeben:

- Für Linux: **`./converter.sh -ds jboss-seam-booking-ds.xml new-datasource-config.xml`**
- Für Microsoft Windows: **`./converter.bat -ds jboss-seam-booking-ds.xml new-datasource-config.xml`**

Beachten Sie, dass der Parameter für die Datenquellenkonversion **`-ds`** lautet.

3. Kopieren Sie das **`<datasource>`**-Element aus der Zielfeile und fügen Sie es unter dem **`<subsystem xmlns="urn:jboss:domain:datasources:1.1"><datasources>`**-Element in die Serverkonfigurationsdatei ein.



## WICHTIG

Sie müssen den Server stoppen, ehe Sie die Server Konfigurations-Datei bearbeiten, damit Ihre Änderungen bei einem Server-Neustart als dauerhaft erstellt werden.

- Falls Sie mit einer Managed Domain arbeiten, so kopieren Sie die XML in die **EAP\_HOME/domain/configuration/domain.xml**-Datei.
  - Falls Sie mit einem Standalone Server arbeiten, kopieren Sie die XML in die **EAP\_HOME/standalone/configuration/standalone.xml**-Datei.
4. Bearbeiten Sie die generierte XML in der neuen Konfigurationsdatei.

Hier ist ein Beispiel der **jboss-seam-booking-ds.xml** Datenquellen-Konfigurationsdatei für das Seam 2.2 Booking-Beispiel, das mit der JBoss EAP 5.x geliefert wurde:

```
<?xml version="1.0" encoding="UTF-8"?>
<datasources>
  <local-tx-datasource>
    <jndi-name>bookingDataSource</jndi-name>
    <connection-url>jdbc:hsqldb:./</connection-url>
    <driver-class>org.hsqldb.jdbcDriver</driver-class>
    <user-name>sa</user-name>
    <password></password>
  </local-tx-datasource>
</datasources>
```

Nachfolgend sehen Sie die Konfigurationsdatei, die durch Ausführung des Converter-Skripts generiert wurde. Die generierte Datei enthält ein **<driver-class>**-Element. Die bevorzugte Weise der Definition der Treiberklasse bei der JBoss EAP 6 ist mittels eines **<driver>**-Elements. Hier ist die resultierende XML in der JBoss EAP 6 Konfigurationsdatei mit Änderungen, um das **<driver-class>**-Element auszukommentieren und das entsprechende **<driver>**-Element hinzuzufügen:

```
<subsystem xmlns="urn:jboss:domain:datasources:1.1">
  <datasources>
    <datasource enabled="true" jndi-
name="java:jboss/datasources/bookingDataSource" jta="true"
      pool-name="bookingDataSource" use-ccm="true" use-java-
context="true">
      <connection-url>jdbc:hsqldb:./</connection-url>
      <!-- Comment out the following driver-class element
           since it is not the preferred way to define this.
      <driver-class>org.hsqldb.jdbcDriver</driver-class>
      -->
      <!-- Specify the driver, which is defined later in the
           datasource -->
      <driver>h2</driver>
      <transaction-isolation>TRANSACTION_NONE</transaction-
isolation>
      <pool>
        <prefill>false</prefill>
        <use-strict-min>false</use-strict-min>
        <flush-strategy>FailingConnectionOnly</flush-strategy>
```

```

</pool>
<security>
  <user-name>sa</user-name>
  <password/>
</security>
<validation>
  <validate-on-match>false</validate-on-match>
  <background-validation>false</background-validation>
  <use-fast-fail>false</use-fast-fail>
</validation>
<timeout/>
<statement>
  <track-statements>false</track-statements>
</statement>
</datasource>
<drivers>
  <!-- The following driver element was not in the
XML target file. It was created manually. -->
  <driver name="h2" module="com.h2database.h2">
    <xa-datasource-class>org.h2.jdbcx.JdbcDataSource</xa-
datasource-class>
  </driver>
</drivers>
</datasources>
</subsystem>

```

[Fehler melden](#)

#### 4.1.9. Verwenden Sie das IronJacamar Migrationstool für die Konvertierung einer Ressourcenadapter-Konfigurationsdatei



##### ANMERKUNG

Das IronJacamar Converter-Skript erfordert Java 7 oder höher.

1. Öffnen Sie eine Befehlszeile und navigieren Sie zum ***IRONJACAMAR\_HOME/docs/as/-*** Verzeichnis.
2. Führen Sie das Converter-Skript durch Eingabe des folgenden Befehls aus:
  - o Für Linux: ***./converter.sh -ra SOURCE\_FILE TARGET\_FILE***
  - o Für Microsoft Windows: ***./converter.bat -ra SOURCE\_FILE TARGET\_FILE***

Die ***SOURCE\_FILE*** ist die Ressourcenadapter -ds.xml-Datei der vorherigen Release. Die ***TARGET\_FILE*** enthält die neue Konfiguration.

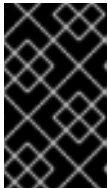
Um zum Beispiel die ***mttestadapter-ds.xml***-Ressourcenadapter-Konfigurationsdatei im aktuellen Verzeichnis zu konvertieren, würden Sie folgendes eingeben:

- o Für Linux: ***./converter.sh -ra mttestadapter-ds.xml new-adapter-config.xml***

- o Für Microsoft Windows: `./converter.bat -ra mttestadapter-ds.xml new-adapter-config.xml`

Beachten Sie, dass der Parameter für die Konvertierung des Ressourcenadapters `-ra` ist.

3. Kopieren Sie das gesamte `<resource-adapters>`-Element aus der Zieldatei und fügen Sie es in der Serverkonfigurationsdatei unter dem `<subsystem xmlns="urn:jboss:domain:resource-adapters:1.1">`-Element ein.



### WICHTIG

Sie müssen den Server stoppen, ehe Sie die Server Konfigurations-Datei bearbeiten, damit Ihre Änderungen bei einem Server-Neustart als dauerhaft erstellt werden.

- o Falls Sie mit einer Managed Domain arbeiten, so kopieren Sie die XML in die `EAP_HOME/domain/configuration/domain.xml`-Datei.
  - o Falls Sie mit einem Standalone Server arbeiten, kopieren Sie die XML in die `EAP_HOME/standalone/configuration/standalone.xml`-Datei.
4. Bearbeiten Sie die generierte XML in der neuen Konfigurationsdatei.

Nachfolgend sehen Sie ein Beispiel für die `mttestadapter-ds.xml`-Ressourcenadapter-Konfigurationsdatei aus der JBoss EAP 5.x TestSuite:

```
<?xml version="1.0" encoding="UTF-8"?>
  <!--
=====
-->
  <!-- ConnectionManager setup for jboss test adapter
-->
  <!-- Build jmx-api (build/build.sh all) and view for config
documentation -->
  <!--
=====
-->
<connection-factories>
  <tx-connection-factory>
    <jndi-name>JBossTestCF</jndi-name>
    <xa-transaction/>
    <rar-name>jbosstestadapter.rar</rar-name>
    <connection-
definition>javax.resource.cci.ConnectionFactory</connection-
definition>
    <config-property name="IntegerProperty"
type="java.lang.Integer">2</config-property>
    <config-property name="BooleanProperty"
type="java.lang.Boolean">>false</config-property>
    <config-property name="DoubleProperty"
type="java.lang.Double">5.5</config-property>
    <config-property name="UrlProperty"
type="java.net.URL">http://www.jboss.org</config-property>
    <config-property name="sleepInStart" type="long">200</config-
property>
```

```

    <config-property name="sleepInStop" type="long">200</config-
property>
  </tx-connection-factory>
  <tx-connection-factory>
    <jndi-name>JBossTestCF2</jndi-name>
    <xa-transaction/>
    <rar-name>jbosstestadapter.rar</rar-name>
    <connection-
definition>javax.resource.cci.ConnectionFactory</connection-
definition>
    <config-property name="IntegerProperty"
type="java.lang.Integer">2</config-property>
    <config-property name="BooleanProperty"
type="java.lang.Boolean">>false</config-property>
    <config-property name="DoubleProperty"
type="java.lang.Double">5.5</config-property>
    <config-property name="UrlProperty"
type="java.net.URL">http://www.jboss.org</config-property>
    <config-property name="sleepInStart" type="long">200</config-
property>
    <config-property name="sleepInStop" type="long">200</config-
property>
  </tx-connection-factory>
  <tx-connection-factory>
    <jndi-name>JBossTestCFByTx</jndi-name>
    <xa-transaction/>
    <track-connection-by-tx>true</track-connection-by-tx>
    <rar-name>jbosstestadapter.rar</rar-name>
    <connection-
definition>javax.resource.cci.ConnectionFactory</connection-
definition>
    <config-property name="IntegerProperty"
type="java.lang.Integer">2</config-property>
    <config-property name="BooleanProperty"
type="java.lang.Boolean">>false</config-property>
    <config-property name="DoubleProperty"
type="java.lang.Double">5.5</config-property>
    <config-property name="UrlProperty"
type="java.net.URL">http://www.jboss.org</config-property>
    <config-property name="sleepInStart" type="long">200</config-
property>
    <config-property name="sleepInStop" type="long">200</config-
property>
  </tx-connection-factory>
</connection-factories>

```

Nachfolgend sehen Sie die Konfigurationsdatei, die durch Ausführung des Converter-Skripts generiert wurde. Ersetzen Sie den Wert des class-name Attributs "FIXME\_MCF\_CLASS\_NAME" in der generierten XML durch den korrekten Klassennamen der gemanagten Connection-Factory, in diesem Fall "org.jboss.test.jca.adapter.TestManagedConnectionFactory". Hier ist die resultierende XML in der JBoss EAP 6 Konfigurationsdatei mit Änderungen am **<class-name>**-Elementwert:

```

<subsystem xmlns="urn:jboss:domain:resource-adapters:1.1">
  <resource-adapters>
    <resource-adapter>

```



```

    <archive>jbosstestadapter.rar</archive>
    <transaction-support>XATransaction</transaction-support>
    <connection-definitions>
      <!-- Replace the "FIXME_MCF_CLASS_NAME" class-name value with the
      correct class name
      <connection-definition class-name="FIXME_MCF_CLASS_NAME"
      enabled="true"
        jndi-name="java:jboss/JBossTestCF" pool-name="JBossTestCF"
        use-ccm="true" use-java-context="true"> -->
    <connection-definition
      class-
name="org.jboss.test.jca.adapter.TestManagedConnectionFactory"
      enabled="true"
      jndi-name="java:jboss/JBossTestCF" pool-name="JBossTestCF"
      use-ccm="true" use-java-context="true">
      <config-property name="IntegerProperty">2</config-property>
      <config-property name="sleepInStart">200</config-property>
      <config-property name="sleepInStop">200</config-property>
      <config-property name="BooleanProperty">>false</config-property>
      <config-property
name="UrlProperty">http://www.jboss.org</config-property>
      <config-property name="DoubleProperty">5.5</config-property>
    </pool>
      <prefill>>false</prefill>
      <use-strict-min>>false</use-strict-min>
      <flush-strategy>FailingConnectionOnly</flush-strategy>
    </pool>
    <security>
      <application/>
    </security>
    <timeout/>
    <validation>
      <background-validation>>false</background-validation>
      <use-fast-fail>>false</use-fast-fail>
    </validation>
  </connection-definition>
</connection-definitions>
</resource-adapter>
<resource-adapter>
  <archive>jbosstestadapter.rar</archive>
  <transaction-support>XATransaction</transaction-support>
  <connection-definitions>
    <!-- Replace the "FIXME_MCF_CLASS_NAME" class-name value with the
    correct class name
    <connection-definition class-name="FIXME_MCF_CLASS_NAME"
    enabled="true"
      jndi-name="java:jboss/JBossTestCF2" pool-name="JBossTestCF2"
      use-ccm="true" use-java-context="true"> -->
  <connection-definition
    class-
name="org.jboss.test.jca.adapter.TestManagedConnectionFactory"
    enabled="true"
    jndi-name="java:jboss/JBossTestCF2" pool-name="JBossTestCF2"
    use-ccm="true" use-java-context="true">
    <config-property name="IntegerProperty">2</config-property>
    <config-property name="sleepInStart">200</config-property>

```



```

    <config-property name="sleepInStop">200</config-property>
    <config-property name="BooleanProperty">>false</config-property>
    <config-property
name="UrlProperty">http://www.jboss.org</config-property>
    <config-property name="DoubleProperty">5.5</config-property>
    <pool>
        <prefill>>false</prefill>
        <use-strict-min>>false</use-strict-min>
        <flush-strategy>FailingConnectionOnly</flush-strategy>
    </pool>
    <security>
        <application/>
    </security>
    <timeout/>
    <validation>
        <background-validation>>false</background-validation>
        <use-fast-fail>>false</use-fast-fail>
    </validation>
</connection-definition>
    </connection-definitions>
</resource-adapter>
<resource-adapter>
    <archive>jbosstestadapter.rar</archive>
    <transaction-support>XATransaction</transaction-support>
    <connection-definitions>
        <!-- Replace the "FIXME_MCF_CLASS_NAME" class-name value with the
correct class name
        <connection-definition class-name="FIXME_MCF_CLASS_NAME"
enabled="true"
            jndi-name="java:jboss/JBossTestCFByTx" pool-
name="JBossTestCFByTx"
            use-ccm="true" use-java-context="true"> -->
    <connection-definition
        class-
name="org.jboss.test.jca.adapter.TestManagedConnectionFactory"
        enabled="true"
        jndi-name="java:jboss/JBossTestCFByTx" pool-
name="JBossTestCFByTx"
        use-ccm="true" use-java-context="true">
        <config-property name="IntegerProperty">2</config-property>
        <config-property name="sleepInStart">200</config-property>
        <config-property name="sleepInStop">200</config-property>
        <config-property name="BooleanProperty">>false</config-property>
        <config-property
name="UrlProperty">http://www.jboss.org</config-property>
        <config-property name="DoubleProperty">5.5</config-property>
        <pool>
            <prefill>>false</prefill>
            <use-strict-min>>false</use-strict-min>
            <flush-strategy>FailingConnectionOnly</flush-strategy>
        </pool>
        <security>
            <application/>
        </security>
        <timeout/>
        <validation>

```

```

        <background-validation>false</background-validation>
        <use-fast-fail>false</use-fast-fail>
    </validation>
</connection-definition>
</connection-definitions>
</resource-adapter>
</resource-adapters>
</subsystem>

```

[Fehler melden](#)

## 4.2. FEHLERBEHEBUNG BEI DER MIGRATION

### 4.2.1. Fehler- und Problembehebung bei der Migration

Aufgrund des Klassenladens, JNDI-Namengebungsregeln und anderen Änderungen im Applikationsserver kann es zu Ausnahmen oder anderen Fehlern kommen, wenn Sie versuchen Ihre Applikation so wie sie ist zu deployen. Nachfolgend erfahren Sie, wie Sie einige der häufiger auftretenden Ausnahmen und Fehler beheben.

- [Abschnitt 4.2.2, »Fehlerbehebung und Auflösung von ClassNotFoundExceptions und NoClassDefFoundErrors«](#)
- [Abschnitt 4.2.5, »Fehler- und Problembehebung von ClassCastExceptions«](#)
- [Abschnitt 4.2.6, »Fehlerbehebung und Auflösung von DuplicateServiceExceptions«](#)
- [Abschnitt 4.2.7, »Fehlerbehebung und Auflösung von JBoss Seam Debug-Seitenfehlern«](#)

[Fehler melden](#)

### 4.2.2. Fehlerbehebung und Auflösung von ClassNotFoundExceptions und NoClassDefFoundErrors

#### Zusammenfassung

ClassNotFoundExceptions-Ausnahmen treten in der Regel wegen einer nicht aufgelösten Abhängigkeit auf. Dies bedeutet, dass Sie die Abhängigkeiten von anderen Modulen explizit definieren oder JARs aus externen Quellen kopieren müssen.

1. Versuchen Sie zuerst, die fehlende Abhängigkeit zu finden. Mehr Einzelheiten dazu finden Sie unter [Abschnitt 4.2.3, »Auffinden der JBoss Modulabhängigkeit«](#)
2. Falls es kein Modul für die fehlende Klasse gibt, suchen Sie das JAR in der vorherigen Installation. Weitere Informationen finden Sie unter [Abschnitt 4.2.4, »Finden Sie das JAR in der vorherigen Installation«](#)

[Fehler melden](#)

### 4.2.3. Auffinden der JBoss Modulabhängigkeit

Um die Abhängigkeit aufzulösen, versuchen Sie zuerst das Modul aufzufinden, das die durch die **ClassNotFoundException** bestimmte Klasse enthält, indem Sie im **EAP\_HOME/modules/system/layers/base/**-Verzeichnis nachsehen. Falls Sie ein Modul für die Klasse finden, so müssen Sie eine Abhängigkeit hinzufügen, um den Eintrag zu manifestieren.

Wenn Sie etwa diese Ausnahme `ClassNotFoundException` im Protokoll sehen:

```
Caused by: java.lang.ClassNotFoundException:
org.apache.commons.logging.Log
    from [Module "deployment.TopicIndex.war:main" from Service Module
Loader]
    at
org.jboss.modules.ModuleClassLoader.findClass(ModuleClassLoader.java:188)
```

Finden Sie das JBoss Modul, das diese Klasse enthält, indem Sie folgendes tun:

#### Prozedur 4.5. Auffinden der Abhängigkeit

1. Bestimmen Sie zuerst, ob ein offensichtliches Modul für die Klasse existiert.
  - a. Navigieren Sie zum **`EAP_HOME/modules/system/layers/base/`**-Verzeichnis und schauen Sie nach dem Modulpfad, der übereinstimmenden Klasse, die in **`ClassNotFoundException`** genannt ist.  
  
Sie finden den Modulpfad **`org/apache/commons/logging/`**.
  - b. Öffnen Sie die **`EAP_HOME/modules/system/layers/base/org/apache/commons/logging/main/module.xml`**-Datei und suchen Sie den Modulnamen. In diesem Fall lautet er **`"org.apache.commons.logging"`**.
  - c. Fügen Sie den Modulnamen zu den Dependencies in der **`MANIFEST.MF`**-Datei hinzu:

```
Manifest-Version: 1.0
Dependencies: org.apache.commons.logging
```
2. Falls es keinen offensichtlichen Modulpfad für die Klasse gibt, so müssen Sie die Abhängigkeit möglicherweise an einem anderen Speicherort suchen.
  - a. Finden Sie die von der **`ClassNotFoundException`** im Tattletale Bericht genannte Klasse.
  - b. Finden Sie das Modul, das das JAR enthält im **`EAP_HOME/modules`**-Verzeichnis und finden Sie den Modulnamen wie im vorherigen Schritt beschrieben.

[Fehler melden](#)

#### 4.2.4. Finden Sie das JAR in der vorherigen Installation

Wird eine Klasse nicht in einem JAR in einem vom Server definierten Modul gefunden, so suchen Sie das JAR in Ihrer **`EAP5_HOME`**-Installation oder dem **`lib/`**-Verzeichnis Ihres früheren Servers.

Wenn Sie etwa diese Ausnahme **`ClassNotFoundException`** im Protokoll sehen:

```
Caused by: java.lang.NoClassDefFoundError:
org.hibernate.validator.ClassValidator at
java.lang.Class.getDeclaredMethods0(Native Method)
```

Finden Sie das JAR, das diese Klasse enthält, indem Sie folgendes tun:

1. Öffnen Sie eine Befehlszeile, und navigieren Sie zum **EAP5\_HOME/-**Verzeichnis.

2. Erteilen Sie den Befehl:

```
grep 'org.hibernate.validator.ClassValidator' `find . \-name '*.jar'`
```

3. Es ist möglich, dass Sie mehr als ein Ergebnis sehen. In diesem Fall ist das folgende Ergebnis das JAR, das wir brauchen:

```
Binary file ./jboss-eap-5.1/seam/lib/hibernate-validator.jar matches
```

4. Kopieren Sie dieses JAR in das **lib/-**Verzeichnis der Applikation.

Falls Sie feststellen, dass Sie eine große Anzahl an JARs benötigen, so ist es vielleicht einfacher, ein Modul für die Klassen zu definieren. Weitere Informationen finden Sie unter *Modules* im Kapitel *Get Started Developing Applications* im *Development Guide* für die JBoss EAP 6 unter

[https://access.redhat.com/site/documentation/JBoss\\_Enterprise\\_Application\\_Platform/](https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/).

5. Erstellen Sie die Applikation neu und deployen Sie diese.

[Fehler melden](#)

#### 4.2.5. Fehler- und Problembehebung von ClassCastExceptions

Es kommt oft zu ClassCastExceptions, da eine Klasse von einem anderen Klassenlader geladen wird als die Klasse, die sie erweitert. Sie können auch auftreten, wenn dieselbe Klasse in mehreren JARs existiert.

1. Durchsuchen Sie die Applikation, um alle JAR(s) zu finden, die die Klasse namens **ClassCastException** enthalten. Falls ein Modul für die Klasse definiert ist, so suchen und entfernen Sie die doppelten JAR(s) aus dem WAR oder EAR der Applikation.
2. Suchen Sie das die Klasse enthaltende JBoss Modul und definieren Sie die Abhängigkeit explizit in der **MANIFEST.MF**-Datei oder in der **jboss-deployment-structure.xml**-Datei. Weitere Informationen finden Sie unter *Class Loading and Subdeployments* im Kapitel *Class Loading and Modules* im *Development Guide* für die JBoss EAP 6 unter [https://access.redhat.com/site/documentation/JBoss\\_Enterprise\\_Application\\_Platform/](https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/).
3. Falls Sie sie nicht mittels der Schritte oben auflösen können, so lässt sich die Ursache des Problems oftmals bestimmen, indem man die Klassenladerinformationen im Protokoll nachsieht. Wenn Sie zum Beispiel die folgende **ClassCastException** im Protokoll sehen:

```
java.lang.ClassCastException: com.example1.CustomClass1 cannot be
cast to com.example2.CustomClass2
```

- a. Drucken Sie in Ihrem Code die Klassenladeinformationen für die von der **ClassCastException** genannten Klassen ins Protokoll, zum Beispiel:

```
logger.info("Class loader for CustomClass1: " +
com.example1.CustomClass1.getClass().getClassLoader().toString())
;
logger.info("Class loader for CustomClass2: " +
```

```
com.example2.CustomClass2.getClass().getClassLoader().toString())
;
```

- b. Die Informationen im Protokoll zeigen, welche Module die Klassen laden und Sie müssen - je nach Ihrer Applikation - in Konflikt stehende JAR(s) entfernen oder verschieben.

[Fehler melden](#)

#### 4.2.6. Fehlerbehebung und Auflösung von DuplicateServiceExceptions

Falls Sie eine DuplicateServiceException für ein Subdeployment eines JAR oder eine Meldung erhalten, die besagt, dass die WAR Applikation bereits installiert ist, wenn Sie Ihr EAR in der JBoss EAP 6 deployen, so kann dies aufgrund der Änderungen der Art und Weise sein, in der JBossWS das Deployment handhabt.

Mit der JBossWS 3.3.0 Release wurde ein neuer Context Root Mapping Algorithmus für Servlet-basierte Endpunkte eingeführt, um eine nahtlose Kompatibilität mit TCK6 zu gewährleisten. Falls das EAR-Archiv der Applikation ein WAR und ein JAR mit demselben Namen enthält, so kann JBossWS einen WAR-Kontext und Web-Kontext mit demselben Namen erstellen. Der Web-Kontext steht im Konflikt mit dem WAR-Kontext, was zu Deployment-Fehlern führt. Lösen Sie die Deployment-Probleme auf eine der folgenden Arten:

- Benennen Sie die JAR-Datei um in einen Namen, der sich vom WAR unterscheidet, damit die generierten Web- und WAR-Kontexte eindeutig sind.
- Liefern Sie ein **<context-root>**-Element in der **jboss-web.xml**-Datei.
- Liefern Sie ein **<context-root>**-Element in der **jboss-webservices.xml**-Datei.
- Passen Sie das **<context-root>**-Element für das WAR in der **application.xml**-Datei an.

[Fehler melden](#)

#### 4.2.7. Fehlerbehebung und Auflösung von JBoss Seam Debug-Seitenfehlern

Nachdem Sie Ihre Applikation migrieren und erfolgreich deployen ist es möglich, dass ein Runtime-Fehler gemeldet wird, der Sie auf die "JBoss Seam Debug"-Seite umleitet. Die URL für diese Seite lautet "http://localhost:8080/APPLICATION\_CONTEXT/debug.seam". Diese Seite gestattet Ihnen die Ansicht und Prüfung der Seam-Komponenten in jedem der Seam-Kontexte, die mit Ihrer aktuellen Login-Sitzung assoziiert werden.

## JBoss Seam Debug Page

This page allows you to browse and inspect components in any of the Seam contexts associated with the current session. It also shows a list of active, long-running conversations. You can select a conversation to view its contents or destroy it.

### Conversations

Conversation ID	Nested?	Activity	Description	View ID	Action
15	false	12:26:58 PM - 12:27:01 PM	Book hotel: Hilton Diagonal Mar	/book.xhtml	<a href="#">Select Destroy</a>

### + Component

### + Conversation Context (None selected)

### + Business Process Context

### + Session Context

### + Application Context

#### Abbildung 4.1. "JBoss Seam Debug"-Seite

Sie wurden wahrscheinlich auf diese Seite umgeleitet, weil Seam eine Ausnahme abgefangen hat, die nicht im Applikationscode gehandhabt wurde. Die Grundursache der Ausnahme kann oft in einem der Links auf der "JBoss Seam Debug Page" gefunden werden.

1. Erweitern Sie den **Component**-Abschnitt auf der Seite und suchen Sie die **`org.jboss.seam.caughtException`**-Komponente.
2. Die Ursache und das Stacktrace should sollte Sie zu den fehlenden Abhängigkeiten führen.

## JBoss Seam Debug Page

This page allows you to browse and inspect components in any of the Seam contexts associated with the current session. It also shows a list of active, long-running conversations. You can select a conversation to view its contents or destroy it.

### Conversations

Conversation ID	Nested?	Activity	Description	View ID	Action
15	false	12:26:58 PM - 12:27:01 PM	Book hotel: Hilton Diagonal Mar	/book.xhtml	<a href="#">Select</a> <a href="#">Destroy</a>

### - Component

Select a component from one of the contexts below

- [Component \(org.jboss.seam.caughtException\)](#)

cause	java.lang.NoClassDefFoundError: org/slf4j/LoggerFactory
class	class javax.servlet.ServletException
localizedMessage	Servlet execution threw an exception
message	Servlet execution threw an exception
rootCause	java.lang.NoClassDefFoundError: org/slf4j/LoggerFactory
stackTrace	[org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:346), org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:248), org.jboss.seam.servlet.SeamFilter\$FilterChainImpl.doFilter(SeamFilter.java:83), org.jboss.seam.web.IdentityFilter.doFilter(IdentityFilter.java:40), [... rest of stacktrace omitted for display purposes]
toString()	javax.servlet.ServletException: Servlet execution threw an exception

+ Conversation Context (None selected)

+ Business Process Context

+ Session Context

+ Application Context

Abbildung 4.2. Komponente `org.jboss.seam.caughtException` Informationen

- Verwenden Sie die in [Abschnitt 4.2.2, »Fehlerbehebung und Auflösung von ClassNotFoundExceptions und NoClassDefFoundErrors«](#) beschriebene Technik zur Auflösung von Abhängigkeiten.

Im Beispiel oben ist die einfachste Lösung das Hinzufügen von **org.slf4j** zum **MANIFEST.MF**

```
Manifest-Version: 1.0
Dependencies: org.slf4j
```

Eine weitere Möglichkeit ist die Hinzufügung einer Abhängigkeit für das Modul zur **jboss-deployment-structure.xml**-Datei:

```
<jboss-deployment-structure>
  <deployment>
    <dependencies>
      <module name="org.slf4j" />
    </dependencies>
  </deployment>
</jboss-deployment-structure>
```

[Fehler melden](#)

## 4.3. ÜBERSICHT DER MIGRATION DER BEISPIELAPPLIKATIONEN

### 4.3.1. Übersicht der Migration der Beispielapplikationen

#### Überblick

Nachfolgend finden Sie eine Liste von Beispielapplikationen der JBoss EAP 5.x, die zur JBoss EAP 6 migriert wurden. Um mehr über Änderungen an einer bestimmten Applikation zu erfahren, klicken Sie auf das Link unten.

- [Abschnitt 4.3.2, »Migration des Seam 2.2 JPA Beispiels zur JBoss EAP 6«](#)
- [Abschnitt 4.3.3, »Migration des Seam 2.2 Buchungsbeispiels zur JBoss EAP 6«](#)
- [Abschnitt 4.3.4, »Migration des Seam 2.2 Buchungsarchivs zur JBoss EAP 6: Schritt-für-Schritt Anleitung«](#)

[Fehler melden](#)

### 4.3.2. Migration des Seam 2.2 JPA Beispiels zur JBoss EAP 6

#### Zusammenfassung

Die folgende Aufgabenliste fasst die Änderungen zusammen, die für die erfolgreiche Migration der Seam 2.2 JPA Beispielanwendung zur JBoss EAP 6 notwendig sind. Sie finden diese Beispielanwendung in der neuesten JBoss EAP 5 Distribution unter **EAP5.x\_HOME/jboss-eap-5.x/seam/examples/jpa/**



#### WICHTIG

Applikationen, die Hibernate direkt mit Seam 2.2 verwenden, können eine innerhalb der Applikation gepackte Version von Hibernate 3 verwenden. Hibernate 4, welches mittels des org.hibernate Moduls der JBoss EAP 6 bereitgestellt wird, wird von Seam 2.2 nicht unterstützt. Dieses Beispiel soll Ihnen dabei helfen, Ihre Applikation auf der JBoss EAP 6 in Betrieb zu nehmen. Bitte beachten Sie, dass das Packen von Hibernate 3 mit einer Seam 2.2 Applikation keine unterstützte Konfiguration ist.

#### Prozedur 4.6. Migration des Seam 2.2 JPA Beispiels

1. **Entfernen Sie die jboss-web.xml-Datei**  
Entfernen Sie die **jboss-web.xml**-Datei aus dem **jboss-seam-jpa.war/WEB-INF/**-Verzeichnis. Das in **jboss-web.xml** definierte Klassenladen ist jetzt das Standardverhalten.
2. **Bearbeiten Sie die jboss-seam-jpa.jar/META-INF/persistence.xml-Datei wie folgt.**
  - a. Entfernen Sie die **hibernate.cache.provider\_class**-Property in der **jboss-seam-jpa.war/WEB-INF/classes/META-INF/persistence.xml**-Datei oder kommentieren Sie sie aus:

```
<!-- <property name="hibernate.cache.provider_class"
value="org.hibernate.cache.HashtableCacheProvider"/> -->
```

- b. Fügen Sie die Provider Modul Property zur **jboss-seam-booking.jar/META-INF/persistence.xml** Datei hinzu:

```
<property name="jboss.as.jpa.providerModule" value="hibernate3-bundled" />
```



- c. Ändern Sie die **jta-data-source** Property, damit sie den standardmäßigen JDBC Datenquellen JNDI Namen benutzt:

```
<jta-data-source>java:jboss/datasources/ExampleDS</jta-data-source>
```

### 3. Fügen Sie die Seam 2.2 Abhängigkeiten hinzu

Kopieren Sie die folgenden JARs aus der Seam 2.2 Distributionsbibliothek, **SEAM\_HOME/lib/** und in das **jboss-seam-jpa.war/WEB-INF/lib/**-Verzeichnis:

- o antlr.jar
- o slf4j-api.jar
- o slf4j-log4j12.jar
- o hibernate-entitymanager.jar
- o hibernate-core.jar
- o hibernate-annotations.jar
- o hibernate-commons-annotations.jar
- o hibernate-validator.jar

### 4. Erstellen Sie eine jboss-deployment-structure-Datei zur Hinzufügung der übrigen Abhängigkeiten

Erstellen Sie eine **jboss-deployment-structure.xml**-Datei im **jboss-seam-jpa.war/WEB-INF/**-Ordner, die die folgenden Daten enthält:

```
<jboss-deployment-structure>
  <deployment>
    <exclusions>
      <module name="javax.faces.api" slot="main"/>
      <module name="com.sun.jsf-impl" slot="main"/>
      <module name="org.hibernate" slot="main"/>
    </exclusions>
    <dependencies>
      <module name="org.apache.log4j" />
      <module name="org.dom4j" />
      <module name="org.apache.commons.logging" />
      <module name="org.apache.commons.collections" />
      <module name="javax.faces.api" slot="1.2"/>
      <module name="com.sun.jsf-impl" slot="1.2"/>
    </dependencies>
  </deployment>
</jboss-deployment-structure>
```

#### Ergebnis:

Die Seam 2.2 JPA Beispielanwendung deployt und läuft nun auf der JBoss EAP 6.

[Fehler melden](#)

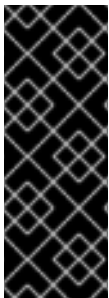
### 4.3.3. Migration des Seam 2.2 Buchungsbeispiels zur JBoss EAP 6

## Zusammenfassung

Die Seam 2.2 Booking EAR Migration ist komplizierter als das Seam 2.2 JPA WAR Beispiel. Die Dokumentation für die Seam 2.2 JPA WAR Beispielmigration finden Sie hier: [Abschnitt 4.3.2, »Migration des Seam 2.2 JPA Beispiels zur JBoss EAP 6«](#). Um die Applikation zu migrieren, müssen Sie das Folgende tun:

1. Initialisieren Sie JSF 1.2 statt des standardmäßigen JSF 2.
2. Bündeln Sie ältere Versionen der Hibernate JARs statt derer, die mit der JBoss EAP 6 geliefert werden.
3. Ändern Sie die JNDI-Bindings, damit diese die neue, portierbare Java EE 6 JNDI Syntax verwenden.

Die ersten beiden Schritte wurden bei der Seam 2.2 JPA WAR-Beispielmigration durchgeführt. Der dritte Schritt ist neu und notwendig, weil das EAR EJBs enthält.



### WICHTIG

Applikationen, die Hibernate direkt mit Seam 2.2 verwenden, können eine innerhalb der Applikation gepackte Version von Hibernate 3 verwenden. Hibernate 4, welches mittels des org.hibernate Moduls der JBoss EAP 6 bereitgestellt wird, wird von Seam 2.2 nicht unterstützt. Dieses Beispiel soll Ihnen dabei helfen, Ihre Applikation auf der JBoss EAP 6 in Betrieb zu nehmen. Bitte beachten Sie, dass das Packen von Hibernate 3 mit einer Seam 2.2 Applikation keine unterstützte Konfiguration ist.

## Prozedur 4.7. Migration des Seam 2.2 Buchungsbeispiels

### 1. Erstellen Sie die `jboss-deployment-structure.xml`-Datei

Erstellen Sie eine neue Datei namens `jboss-deployment-structure.xml` im `jboss-seam-booking.ear/META-INF/` und fügen Sie folgenden Inhalt hinzu:

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-
structure:1.0">
  <deployment>
    <dependencies>
      <module name="javax.faces.api" slot="1.2" export="true"/>
      <module name="com.sun.jsf-impl" slot="1.2"
export="true"/>
      <module name="org.apache.log4j" export="true"/>
      <module name="org.dom4j" export="true"/>
      <module name="org.apache.commons.logging" export="true"/>
      <module name="org.apache.commons.collections"
export="true"/>
    </dependencies>
    <exclusions>
      <module name="org.hibernate" slot="main"/>
    </exclusions>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
      <module name="javax.faces.api" slot="main"/>
      <module name="com.sun.jsf-impl" slot="main"/>
    </exclusions>
  </sub-deployment>
</jboss-deployment-structure>
```

```

        <dependencies>
          <module name="javax.faces.api" slot="1.2"/>
          <module name="com.sun.jsf-impl" slot="1.2"/>
        </dependencies>
      </sub-deployment>
    </jboss-deployment-structure>

```

## 2. Ändern Sie die **jboss-seam-booking.jar/META-INF/persistence.xml**-Datei wie folgt.

- a. Entfernen Sie die Hibernate-Property für die Cache Provider-Klasse oder kommentieren Sie sie aus:

```

<!-- <property name="hibernate.cache.provider_class"
value="org.hibernate.cache.HashtableCacheProvider"/> -->

```

- b. Provider-Moduleeigenschaft zur **jboss-seam-booking.jar/META-INF/persistence.xml** Datei hinzufügen:

```

<property name="jboss.as.jpa.providerModule" value="hibernate3-
bundled" />

```

- c. Die **jta-data-source** Eigenschaft ändern um den standardmäßigen JDBC Datenquellen JNDI Namen zu benutzen:

```

<jta-data-source>java:jboss/datasources/ExampleDS</jta-data-
source>

```

## 3. Kopieren Sie JARs aus der Seam 2.2 Distribution

Kopieren Sie die folgenden JARs aus der Seam 2.2 Distribution **EAP5.x\_HOME/jboss-eap5.x/seam/lib/** in das **jboss-seam-booking.ear/lib**-Verzeichnis.

```

antlr.jar
slf4j-api.jar
slf4j-log4j12.jar
hibernate-core.jar
hibernate-entitymanager.jar
hibernate-validator.jar
hibernate-annotations.jar
hibernate-commons-annotations.jar

```

## 4. Ändern Sie die JNDI Lookup-Namen

Ändern Sie die JNDI Lookup-Strings in der **jboss-seam-booking.war/WEB-INF/components.xml**-Datei. Wegen der neuen JNDI portierbaren Regeln bindet die JBoss EAP 6 jetzt EJBs mittels JNDI portierbarer Syntax-Regeln und Sie nicht das einzelne `jndiPattern` verwenden wie noch in der JBoss EAP 5. Dies ist, wie Sie die Applikations EJB JNDI Lookup-Strings in der JBoss EAP 6 ändern müssen:

```

java:global/jboss-seam-booking/jboss-seam-
booking/HotelSearchingAction!org.jboss.seam.example.booking.HotelSea
rching
java:app/jboss-seam-
booking/HotelSearchingAction!org.jboss.seam.example.booking.HotelSea
rching

```

```

java:module/HotelSearchingAction!org.jboss.seam.example.booking.HotelSearching
java:global/jboss-seam-booking/jboss-seam-booking/HotelSearchingAction
java:app/jboss-seam-booking/HotelSearchingAction
java:module/HotelSearchingAction

```

Die JNDI-Lookup-Strings für die Seam 2.2 Framework EJBs müssen wie folgt geändert werden:

```

java:global/jboss-seam-booking/jboss-seam/EjbSynchronizations!org.jboss.seam.transaction.LocalEjbSynchronizations
java:app/jboss-seam/EjbSynchronizations!org.jboss.seam.transaction.LocalEjbSynchronizations
java:module/EjbSynchronizations!org.jboss.seam.transaction.LocalEjbSynchronizations
java:global/jboss-seam-booking/jboss-seam/EjbSynchronizations
java:app/jboss-seam/EjbSynchronizations
java:module/EjbSynchronizations

```

Wählen Sie eine der folgenden Vorgehensweisen:

a. **Fügen Sie Komponentenelemente hinzu**

Sie können einen **jndi-name** für jedes EJB der **WEB-INF/components.xml** hinzufügen:

```

<component
class="org.jboss.seam.transaction.EjbSynchronizations" jndi-
name="java:app/jboss-seam/EjbSynchronizations"/>
<component
class="org.jboss.seam.async.TimerServiceDispatcher" jndi-
name="java:app/jboss-seam/TimerServiceDispatcher"/>
<component
class="org.jboss.seam.example.booking.AuthenticatorAction" jndi-
name="java:app/jboss-seam-booking/AuthenticatorAction" />
<component
class="org.jboss.seam.example.booking.BookingListAction" jndi-
name="java:app/jboss-seam-booking/BookingListAction" />
<component
class="org.jboss.seam.example.booking.RegisterAction" jndi-
name="java:app/jboss-seam-booking/RegisterAction" />
<component
class="org.jboss.seam.example.booking.HotelSearchingAction" jndi-
name="java:app/jboss-seam-booking/HotelSearchingAction" />
<component
class="org.jboss.seam.example.booking.HotelBookingAction" jndi-
name="java:app/jboss-seam-booking/HotelBookingAction" />
<component
class="org.jboss.seam.example.booking.ChangePasswordAction" jndi-
name="java:app/jboss-seam-booking/ChangePasswordAction" />

```

- b. Sie können den Code bearbeiten, indem Sie die **@JNDIName(value="")**-Annotation hinzufügen, die den JNDI-Pfad festlegt. Ein Beispiel für den veränderten stateless Session Bean Code sehen Sie unten. Eine ausführliche Beschreibung dieses Vorgangs finden Sie in der Seam 2.2 Referenzdokumentation.

```

@Stateless
@Name("authenticator")
@JndiName(value="java:app/jboss-seam-
booking/AuthenticatorAction")
public class AuthenticatorAction
    implements Authenticator
{
    ...
}

```

**Ergebnis:**

Die Seam 2.2 JPA Buchungsanwendung deployt und läuft nun auf der JBoss EAP 6.

[Fehler melden](#)

#### 4.3.4. Migration des Seam 2.2 Buchungsarchivs zur JBoss EAP 6: Schritt-für-Schritt Anleitung

Dies ist eine Schritt-für-Schritt Anleitung für die Migration des Seam 2.2 Booking Applikationsarchivs von der JBoss EAP 5.X zur JBoss EAP 6. Obwohl es bessere Vorgehensweise für die Migration von Anwendungen gibt, sind viele Entwickler möglicherweise versucht, das Applikationsarchiv so wie es ist zum JBoss EAP 6 Server zu deployen und zu sehen, was passiert. Dieses Dokument will Ihnen zeigen, mit welchen Problemen Sie rechnen müssen, wenn Sie dies tun und wie Sie diese lösen können.

Für dieses Beispiel wird das Applikations-EAR in das **EAP6\_HOME/standalone/deployments-**Verzeichnis deployt, ohne das irgendwelche Änderungen vorgenommen werden, außer der Extraktion der Archive. Dies gestattet es Ihnen, die innerhalb der Archive befindlichen XML-Dateien leicht zu bearbeiten, wenn Probleme auftreten, die Sie lösen müssen.

**WICHTIG**

Applikationen, die Hibernate direkt mit Seam 2.2 verwenden, können eine innerhalb der Applikation gepackte Version von Hibernate 3 verwenden. Hibernate 4, welches mittels des org.hibernate Moduls der JBoss EAP 6 bereitgestellt wird, wird von Seam 2.2 nicht unterstützt. Dieses Beispiel soll Ihnen dabei helfen, Ihre Applikation auf der JBoss EAP 6 in Betrieb zu nehmen. Bitte beachten Sie, dass das Packen von Hibernate 3 mit einer Seam 2.2 Applikation keine unterstützte Konfiguration ist.

**Prozedur 4.8. Migrieren Sie die Applikation**

1. [Abschnitt 4.3.5, »Erstellen und Deployment der JBoss EAP 5.X Version der Seam 2.2 Booking Applikation«](#)
2. [Abschnitt 4.3.6, »Fehlerbehebung und Auflösung von Seam 2.2 Booking Archive Deployment Fehlern und Ausnahmen«](#)
3. [Abschnitt 4.3.7, »Fehlerbehebung und Auflösung von Seam 2.2 Booking Archive Runtime Fehlern und Ausnahmen«](#)

An diesem Punkt können Sie auf die Applikation in einem Browser mittels der URL <http://localhost:8080/seam-booking/> zugreifen. Melden Sie sich mit demo/demo an und es erscheint die Booking-Begrüßungsseite.

**Übersicht der Änderungen**

[Abschnitt 4.3.8, »Übersicht über eine Zusammenfassung von Änderungen bei der Migration der Seam 2.2 Booking Application«](#)

[Fehler melden](#)

### 4.3.5. Erstellen und Deployment der JBoss EAP 5.X Version der Seam 2.2 Booking Applikation

Ehe Sie diese Applikation migrieren, müssen Sie die JBoss EAP 5.X Seam 2.2 Booking Applikation erstellen, das Archiv extrahieren und es in den JBoss EAP 6 Deployment Ordner kopieren.

#### Prozedur 4.9. Erstellen und Deployment des EAR

1. Erstellen Sie das EAR:

```
$ cd /EAP5_HOME/jboss-eap5.x/seam/examples/booking
$ ANT_HOME/ant explode
```

Ersetzen Sie *jboss-eap5.x* durch die Version der JBoss EAP, von der Sie migrieren.

2. Kopieren Sie das EAR in das *EAP6\_HOME* Deployments Verzeichnis:

```
$ cp -r EAP5_HOME/seam/examples/booking/exploded-archives/jboss-seam-booking.ear EAP6_HOME/standalone/deployments/
$ cp -r EAP5_HOME/seam/examples/booking/exploded-archives/jboss-seam-booking.war EAP6_HOME/standalone/deployments/jboss-seam.ear
$ cp -r EAP5_HOME/seam/examples/booking/exploded-archives/jboss-seam-booking.jar EAP6_HOME/standalone/deployments/jboss-seam.ear
```

3. Starten Sie den JBoss EAP 6 Server und prüfen Sie das Protokoll. Sie sehen:

```
INFO [org.jboss.as.deployment] (DeploymentScanner-threads - 1) Found
jboss-seam-booking.ear in deployment directory.
    To trigger deployment create a file called jboss-seam-
booking.ear.dodeploy
```

4. Erstellen Sie eine leere Datei namens **jboss-seam-booking.ear.dodeploy** und kopieren Sie sie in das **EAP6\_HOME/standalone/deployments**-Verzeichnis. Sie müssen diese Datei viele Male in das Deployments-Verzeichnis kopieren, während Sie diese Applikation migrieren, daher sollten Sie sie an einem Speicherort verwahren, an dem Sie sie leicht finden. Im Protokoll sollten Sie jetzt die folgenden Nachrichten sehen, die Ihnen anzeigen, dass sie deployt wird:

```
INFO [org.jboss.as.server.deployment] (MSC service thread 1-1)
Starting deployment of "jboss-seam-booking.ear"
INFO [org.jboss.as.server.deployment] (MSC service thread 1-3)
Starting deployment of "jboss-seam-booking.jar"
INFO [org.jboss.as.server.deployment] (MSC service thread 1-6)
Starting deployment of "jboss-seam.jar"
INFO [org.jboss.as.server.deployment] (MSC service thread 1-2)
Starting deployment of "jboss-seam-booking.war"
```

An diesem Punkt tritt der erste Deployment-Fehler auf. Im nächsten Schritt gehen Sie auf jedes Problem einzeln ein und lernen es zu beheben.

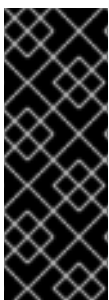
Um mehr zur Fehlerbehebung und der Auflösung von Deployment-Problemen zu erfahren, klicken Sie hier: [Abschnitt 4.3.6, »Fehlerbehebung und Auflösung von Seam 2.2 Booking Archive Deployment Fehlern und Ausnahmen«](#)

Um zum vorherigen Thema zurückzukehren, klicken Sie hier: [Abschnitt 4.3.4, »Migration des Seam 2.2 Buchungsarchivs zur JBoss EAP 6: Schritt-für-Schritt Anleitung«](#)

[Fehler melden](#)

### 4.3.6. Fehlerbehebung und Auflösung von Seam 2.2 Booking Archive Deployment Fehlern und Ausnahmen

Im vorherigen Schritt [Abschnitt 4.3.5, »Erstellen und Deployment der JBoss EAP 5.X Version der Seam 2.2 Booking Applikation«](#) haben Sie die JBoss EAP 5.X Seam 2.2 Booking Applikation erstellt und Sie zum JBoss EAP 6 Deployment-Ordner deployt. In diesem Schritt beheben Sie Fehler und lösen jeden Deployment-Fehler auf, auf den Sie treffen.



#### WICHTIG

Applikationen, die Hibernate direkt mit Seam 2.2 verwenden, können eine innerhalb der Applikation gepackte Version von Hibernate 3 verwenden. Hibernate 4, welches mittels des org.hibernate Moduls der JBoss EAP 6 bereitgestellt wird, wird von Seam 2.2 nicht unterstützt. Dieses Beispiel soll Ihnen dabei helfen, Ihre Applikation auf der JBoss EAP 6 in Betrieb zu nehmen. Bitte beachten Sie, dass das Packen von Hibernate 3 mit einer Seam 2.2 Applikation keine unterstützte Konfiguration ist.

### Prozedur 4.10. Fehlerbehebung und Auflösung von Deployment-Fehlern und Ausnahmen

1. Problem - java.lang.ClassNotFoundException: javax.faces.FacesException

Wenn Sie die Applikation deployen, so enthält das Protokoll den folgenden Fehler:

```
ERROR \[org.jboss.msc.service.fail\] (MSC service thread 1-1)
MSC00001: Failed to start service jboss.deployment.subunit."jboss-
seam-booking.ear"."jboss-seam-booking.war".POST_MODULE:
org.jboss.msc.service.StartException in service
jboss.deployment.subunit."jboss-seam-booking.ear"."jboss-seam-
booking.war".POST_MODULE:
Failed to process phase POST_MODULE of subdeployment "jboss-seam-
booking.war" of deployment "jboss-seam-booking.ear"
(.. additional logs removed ...)
Caused by: java.lang.ClassNotFoundException:
javax.faces.FacesException from \[Module "deployment.jboss-seam-
booking.ear:main" from Service Module Loader\]
at
org.jboss.modules.ModuleClassLoader.findClass(ModuleClassLoader.java
:191)
```

#### Was es bedeutet:

Die ClassNotFoundException zeigt eine fehlende Abhängigkeit an. In diesem Fall kann sie die Klasse **javax.faces.FacesException** nicht finden, und Sie müssen die Abhängigkeit ausdrücklich hinzufügen.

#### Um das Problem zu beseitigen:



Suchen Sie den Modulnamen für diese Klasse im **EAP6\_HOME/modules/system/layers/base/-**Verzeichnis, indem Sie nach einem Pfad suchen, der mit der fehlenden Klasse übereinstimmt. In diesem Fall finden Sie 2 übereinstimmende Module:

```
javax/faces/api/main
javax/faces/api/1.2
```

Beide Module besitzen denselben Modulnamen: **javax.faces.api** aber eines im Hauptverzeichnis ist für SF 2.0 und das im 1.2-Verzeichnis ist für JSF 1.2. Falls nur ein Modul verfügbar wäre, so könnten Sie einfach eine **MANIFEST.MF**-Datei erstellen und die Modulabhängigkeit hinzufügen. Aber in diesem Fall wollen Sie die JSF 1.2 Version und nicht die 2.0 Version im Hauptverzeichnis verwenden, weshalb Sie eine festlegen und die andere ausschließen müssen. Um dies zu tun, erstellen Sie eine **jboss-deployment-structure.xml**-Datei im **META-INF/-**Verzeichnis des EAR, die die folgenden Daten enthält:

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-
structure:1.0">
  <deployment>
    <dependencies>
      <module name="javax.faces.api" slot="1.2" export="true"/>
    </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
      <module name="javax.faces.api" slot="main"/>
    </exclusions>
    <dependencies>
      <module name="javax.faces.api" slot="1.2"/>
    </dependencies>
  </sub-deployment>
</jboss-deployment-structure>
```

Im **deployment**-Abschnitt fügen Sie die Abhängigkeit für das **javax.faces.api** für das JSF 1.2 Modul hinzu. Sie fügen außerdem die Abhängigkeit für das JSF 1.2 Modul im Subdeployment-Abschnitt für das WAR hinzu und schließen das Modul für JSF 2.0 aus.

Deployen Sie die Applikation, indem Sie die **EAP6\_HOME/standalone/deployments/jboss-seam-booking.ear.failed**-Datei löschen und eine leere **jboss-seam-booking.ear.dodeploy**-Datei in demselben Verzeichnis erstellen.

## 2. Problem - java.lang.ClassNotFoundException: org.apache.commons.logging.Log

Wenn Sie die Applikation deployen, so enthält das Protokoll den folgenden Fehler:

```
ERROR [org.jboss.msc.service.fail] (MSC service thread 1-8)
MSC00001: Failed to start service jboss.deployment.unit."jboss-seam-
booking.ear".INSTALL:
org.jboss.msc.service.StartException in service
jboss.deployment.unit."jboss-seam-booking.ear".INSTALL:
Failed to process phase INSTALL of deployment "jboss-seam-
booking.ear"
(.. additional logs removed ...)
```



```
Caused by: java.lang.ClassNotFoundException:
org.apache.commons.logging.Log from [Module "deployment.jboss-seam-
booking.ear.jboss-seam-booking.war:main" from Service Module Loader]
```

### Was es bedeutet:

Die **ClassNotFoundException** zeigt eine fehlende Abhängigkeit an. In diesem Fall kann sie die Klasse **org.apache.commons.logging.Log** nicht finden, und Sie müssen die Abhängigkeit ausdrücklich hinzufügen.

### Um das Problem zu beseitigen:

Suchen Sie den Modulnamen für diese Klasse im

**EAP6\_HOME/modules/system/layers/base/-**Verzeichnis, indem Sie nach einem Pfad suchen, der mit der fehlenden Klasse übereinstimmt. In diesem Fall finden Sie ein Modul, das mit dem Pfad **org/apache/commons/logging/** übereinstimmt. Der Modulname lautet "org.apache.commons.logging".

Bearbeiten Sie die **jboss-deployment-structure.xml**-Datei, um die Modulabhängigkeit zum Deployment-Abschnitt der Datei hinzuzufügen.

```
<module name="org.apache.commons.logging" export="true"/>
```

Die **jboss-deployment-structure.xml** sollte nun wie folgt aussehen:

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-
structure:1.0">
  <deployment>
    <dependencies>
      <module name="javax.faces.api" slot="1.2" export="true"/>
      <module name="org.apache.commons.logging" export="true"/>
    </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
      <module name="javax.faces.api" slot="main"/>
    </exclusions>
    <dependencies>
      <module name="javax.faces.api" slot="1.2"/>
    </dependencies>
  </sub-deployment>
</jboss-deployment-structure>
```

Deployen Sie die Applikation, indem Sie die

**EAP6\_HOME/standalone/deployments/jboss-seam-booking.ear.failed**-Datei löschen und eine leere **jboss-seam-booking.ear.dodeploy**-Datei in demselben Verzeichnis erstellen.

### 3. Problem - java.lang.ClassNotFoundException: org.dom4j.DocumentException

Wenn Sie die Applikation deployen, so enthält das Protokoll den folgenden Fehler:

```
ERROR [org.apache.catalina.core.ContainerBase.[jboss.web].[default-
host].[/seam-booking]] (MSC service thread 1-3) Exception sending
context initialized event to listener instance of class
org.jboss.seam.servlet.SeamListener: java.lang.NoClassDefFoundError:
```

```
org/dom4j/DocumentException
(... additional logs removed ...)
Caused by: java.lang.ClassNotFoundException:
org.dom4j.DocumentException from [Module "deployment.jboss-seam-
booking.ear.jboss-seam.jar:main" from Service Module Loader]
```

### Was es bedeutet:

Die **ClassNotFoundException** zeigt eine fehlende Abhängigkeit an. In diesem Fall kann sie die Klasse **org.dom4j.DocumentException** nicht finden.

### Um das Problem zu beseitigen:

Suchen Sie den Modulnamen im **EAP6\_HOME/modules/system/layers/base/-** Verzeichnis, indem Sie nach der **org/dom4j/DocumentException** suchen. Der Modulname ist "org.dom4j". Bearbeiten Sie die **jboss-deployment-structure.xml**-Datei, um die Modulabhängigkeit zum Deployment-Abschnitt der Datei hinzuzufügen.

```
<module name="org.dom4j" export="true"/>
```

Die **jboss-deployment-structure.xml**-Datei sollte nun wie folgt aussehen:

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-
structure:1.0">
  <deployment>
    <dependencies>
      <module name="javax.faces.api" slot="1.2" export="true"/>
      <module name="org.apache.commons.logging" export="true"/>
      <module name="org.dom4j" export="true"/>
    </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
      <module name="javax.faces.api" slot="main"/>
    </exclusions>
    <dependencies>
      <module name="javax.faces.api" slot="1.2"/>
    </dependencies>
  </sub-deployment>
</jboss-deployment-structure>
```

Deployen Sie die Applikation, indem Sie die **EAP6\_HOME/standalone/deployments/jboss-seam-booking.ear.failed**-Datei löschen und eine leere **jboss-seam-booking.ear.dodeploy**-Datei in demselben Verzeichnis erstellen.

#### 4. Problem - java.lang.ClassNotFoundException: org.hibernate.validator.InvalidValue

Wenn Sie die Applikation deployen, so enthält das Protokoll den folgenden Fehler:

```
ERROR [org.apache.catalina.core.ContainerBase.[jboss.web].[default-
host].[/seam-booking]] (MSC service thread 1-6) Exception sending
context initialized event to listener instance of class
org.jboss.seam.servlet.SeamListener: java.lang.RuntimeException:
Could not create Component:
org.jboss.seam.international.statusMessages
```

```
(... additional logs removed ...)
Caused by: java.lang.ClassNotFoundException:
org.hibernate.validator.InvalidValue from [Module "deployment.jboss-
seam-booking.ear.jboss-seam.jar:main" from Service Module Loader]
```

### Was es bedeutet:

Die **ClassNotFoundException** zeigt eine fehlende Abhängigkeit an. In diesem Fall kann sie die Klasse **org.hibernate.validator.InvalidValue** nicht finden.

### Um das Problem zu beseitigen:

Es gibt ein Modul für "org.hibernate.validator", aber das JAR enthält die **org.hibernate.validator.InvalidValue**-Klasse nicht, so dass das Hinzufügen der Modulabhängigkeit das Problem nicht behebt. In diesem Fall war das die Klasse enthaltende JAR Teil des JBoss EAP 5.X Deployments. Suchen Sie das JAR, das die fehlende Klasse im **EAP5\_HOME/seam/lib**-Verzeichnis enthält. Um dies zu tun öffnen Sie eine Konsole und geben Sie das folgende ein:

```
$ cd EAP5_HOME/seam/lib
$ grep 'org.hibernate.validator.InvalidValue' `find . -name '*.jar'`
```

Das Ergebnis zeigt:

```
$ Binary file ./hibernate-validator.jar matches
$ Binary file ./test/hibernate-all.jar matches
```

In diesem Fall kopieren Sie das **hibernate-validator.jar** in das **jboss-seam-booking.ear/lib**-Verzeichnis:

```
$ cp EAP5_HOME/seam/lib/hibernate-validator.jar jboss-seam-
booking.ear/lib
```

Deployen Sie die Applikation, indem Sie die **EAP6\_HOME/standalone/deployments/jboss-seam-booking.ear.failed**-Datei löschen und eine leere **jboss-seam-booking.ear.dodeploy**-Datei in demselben Verzeichnis erstellen.

## 5. Problem - java.lang.InstantiationException: org.jboss.seam.jsf.SeamApplicationFactory

Wenn Sie die Applikation deployen, so enthält das Protokoll den folgenden Fehler:

```
INFO [javax.enterprise.resource.webcontainer.jsf.config] (MSC
service thread 1-7) Unsanitized stacktrace from failed start...:
com.sun.faces.config.ConfigurationException: Factory
'javax.faces.application.ApplicationFactory' was not configured
properly.
    at
com.sun.faces.config.processor.FactoryConfigProcessor.verifyFactorie
sExist(FactoryConfigProcessor.java:296) [jsf-impl-2.0.4-b09-
jbossorg-4.jar:2.0.4-b09-jbossorg-4]
    (... additional logs removed ...)
Caused by: javax.faces.FacesException:
org.jboss.seam.jsf.SeamApplicationFactory
    at
```

```

javax.faces.FactoryFinder.getImplGivenPreviousImpl(FactoryFinder.java:606) [jsf-api-1.2_13.jar:1.2_13-b01-FCS]
    (... additional logs removed ...)
    at
com.sun.faces.config.processor.FactoryConfigProcessor.verifyFactoryExist(FactoryConfigProcessor.java:294) [jsf-impl-2.0.4-b09-jbossorg-4.jar:2.0.4-b09-jbossorg-4]
    ... 11 more
Caused by: java.lang.InstantiationException:
org.jboss.seam.jsf.SeamApplicationFactory
    at java.lang.Class.newInstance0(Class.java:340) [:1.6.0_25]
    at java.lang.Class.newInstance(Class.java:308) [:1.6.0_25]
    at
javax.faces.FactoryFinder.getImplGivenPreviousImpl(FactoryFinder.java:604) [jsf-api-1.2_13.jar:1.2_13-b01-FCS]
    ... 16 more

```

### Was es bedeutet:

Die **com.sun.faces.config.ConfigurationException** und **java.lang.InstantiationException** zeigen ein Abhängigkeitenproblem an. In diesem Fall ist die Ursache nicht offenkundig.

### Um das Problem zu beseitigen:

Sie müssen das Modul finden, das die **com.sun.faces**-Klassen enthält. Es gibt zwar kein **com.sun.faces**-Modul, aber es gibt zwei **com.sun.jsf-impl**-Module. Eine schnelle Prüfung des **jsf-impl-1.2\_13.jar** im 1.2-Verzeichnis zeigt die **com.sun.faces**-Klassen. Wie bei der **javax.faces.FacesException ClassNotFoundException** wollen Sie die JSF 1.2 Version und nicht die JSF 2.0 Version im Hauptverzeichnis, daher müssen Sie eines festlegen und das andere ausschließen. Sie müssen die **jboss-deployment-structure.xml** bearbeiten, um die um die Modulabhängigkeit zum Deployment-Abschnitt der Datei hinzuzufügen. Sie müssen es auch zum WAR-Subdeployment hinzufügen und das JSF 2.0 Modul ausschließen. Die Datei sollte jetzt so aussehen:

```

<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.0">
  <deployment>
    <dependencies>
      <module name="javax.faces.api" slot="1.2" export="true"/>
      <module name="com.sun.jsf-impl" slot="1.2" export="true"/>
      <module name="org.apache.commons.logging" export="true"/>
      <module name="org.dom4j" export="true"/>
    </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">
    <exclusions>
      <module name="javax.faces.api" slot="main"/>
      <module name="com.sun.jsf-impl" slot="main"/>
    </exclusions>
    <dependencies>
      <module name="javax.faces.api" slot="1.2"/>
      <module name="com.sun.jsf-impl" slot="1.2"/>
    </dependencies>
  </sub-deployment>
</jboss-deployment-structure>

```

```

        </dependencies>
    </sub-deployment>
</jboss-deployment-structure>

```

Deployen Sie die Applikation, indem Sie die **EAP6\_HOME/standalone/deployments/jboss-seam-booking.ear.failed**-Datei löschen und eine leere **jboss-seam-booking.ear.dodeploy**-Datei in demselben Verzeichnis erstellen.

#### 6. Problem - java.lang.ClassNotFoundException: org.apache.commons.collections.ArrayStack

Wenn Sie die Applikation deployen, so enthält das Protokoll den folgenden Fehler:

```

ERROR [org.apache.catalina.core.ContainerBase.[jboss.web].[default-
host].[/seam-booking]] (MSC service thread 1-1) Exception sending
context initialized event to listener instance of class
com.sun.faces.config.ConfigureListener: java.lang.RuntimeException:
com.sun.faces.config.ConfigurationException: CONFIGURATION FAILED!
org.apache.commons.collections.ArrayStack from [Module
"deployment.jboss-seam-booking.ear:main" from Service Module Loader]
(... additional logs removed ...)
Caused by: java.lang.ClassNotFoundException:
org.apache.commons.collections.ArrayStack from [Module
"deployment.jboss-seam-booking.ear:main" from Service Module Loader]

```

#### Was es bedeutet:

Die **ClassNotFoundException** zeigt eine fehlende Abhängigkeit an. In diesem Fall kann sie die Klasse **org.apache.commons.collections.ArrayStack** nicht finden.

#### Um das Problem zu beseitigen:

Suchen Sie den Modulnamen im **EAP6\_HOME/modules/system/layers/base/-** Verzeichnis, indem Sie nach dem **org/apache/commons/collections**-Pfad suchen. Der Modulname lautet "org.apache.commons.collections". Bearbeiten Sie die **jboss-deployment-structure.xml**, um dem Deployment-Abschnitt der Datei die Modulabhängigkeit hinzuzufügen.

```

<module name="org.apache.commons.collections" export="true"/>

```

Die **jboss-deployment-structure.xml**-Datei sollte nun wie folgt aussehen:

```

<jboss-deployment-structure xmlns="urn:jboss:deployment-
structure:1.0">
  <deployment>
    <dependencies>
      <module name="javax.faces.api" slot="1.2" export="true"/>
      <module name="com.sun.jsf-impl" slot="1.2"
export="true"/>
      <module name="org.apache.commons.logging" export="true"/>
      <module name="org.dom4j" export="true"/>
      <module name="org.apache.commons.collections"
export="true"/>
    </dependencies>
  </deployment>
  <sub-deployment name="jboss-seam-booking.war">

```

```

<exclusions>
  <module name="javax.faces.api" slot="main"/>
  <module name="com.sun.jsf-impl" slot="main"/>
</exclusions>
<dependencies>
  <module name="javax.faces.api" slot="1.2"/>
  <module name="com.sun.jsf-impl" slot="1.2"/>
</dependencies>
</sub-deployment>
</jboss-deployment-structure>

```

Deployen Sie die Applikation, indem Sie die **EAP6\_HOME/standalone/deployments/jboss-seam-booking.ear.failed**-Datei löschen und eine leere **jboss-seam-booking.ear.dodeploy**-Datei in demselben Verzeichnis erstellen.

## 7. Problem - Dienste mit fehlenden/nicht verfügbaren Abhängigkeiten

Wenn Sie die Applikation deployen, so enthält das Protokoll den folgenden Fehler:

```

ERROR [org.jboss.as.deployment] (DeploymentScanner-threads - 2)
{"Composite operation failed and was rolled back. Steps that
failed:" => {"Operation step-2" => {"Services with
missing/unavailable dependencies" =>
["jboss.deployment.subunit.\"jboss-seam-booking.ear\".\"jboss-seam-
booking.jar\".component.AuthenticatorAction.START missing [
jboss.naming.context.java.comp.jboss-seam-booking.\"jboss-seam-
booking.jar\".AuthenticatorAction.\"env/org.jboss.seam.example.booki
ng.AuthenticatorAction/em\" ]\", \"jboss.deployment.subunit.\"jboss-
seam-booking.ear\".\"jboss-seam-
booking.jar\".component.HotelSearchingAction.START missing [
jboss.naming.context.java.comp.jboss-seam-booking.\"jboss-seam-
booking.jar\".HotelSearchingAction.\"env/org.jboss.seam.example.book
ing.HotelSearchingAction/em\" ]\", \"
(... additional logs removed ...)
\"jboss.deployment.subunit.\"jboss-seam-booking.ear\".\"jboss-seam-
booking.jar\".component.BookingListAction.START missing [
jboss.naming.context.java.comp.jboss-seam-booking.\"jboss-seam-
booking.jar\".BookingListAction.\"env/org.jboss.seam.example.booking
.BookingListAction/em\" ]\", \"jboss.persistenceunit.\"jboss-seam-
booking.ear/jboss-seam-booking.jar#bookingDatabase\" missing [
jboss.naming.context.java.bookingDatasource ]"]}}}

```

### Was es bedeutet:

Wenn Sie einen “Services with missing/unavailable dependencies”-Fehler erhalten, sehen Sie sich den Text innerhalb der Klammern nach “missing” an. In diesem Fall sehen Sie:

```

missing [ jboss.naming.context.java.comp.jboss-seam-booking.\"jboss-
seam-
booking.jar\".AuthenticatorAction.\"env/org.jboss.seam.example.booki
ng.AuthenticatorAction/em\" ]

```

Das “/em” steht für ein Problem mit dem Entity Manager und der Datenquelle.

### Um das Problem zu beseitigen:

Bei der JBoss EAP 6 wurde die Datenquellenkonfiguration geändert und muss in der **EAP6\_HOME/standalone/configuration/standalone.xml**-Datei definiert werden. Da die JBoss EAP 6 mit einer bereits in der **standalone.xml**-Datei definierten Datenbank geliefert wird, bearbeiten Sie die **persistence.xml**-Datei so, dass Sie die Beispieldatenbank in dieser Applikation verwendet. Wenn Sie sich die **standalone.xml**-Datei ansehen, so sehen Sie, dass der **jndi-name** für die Beispieldatenbank **java:jboss/datasources/ExampleDS** ist. Bearbeiten Sie die **jboss-seam-booking.jar/META-INF/persistence.xml**-Datei, um das bestehende **jta-data-source**-Element auszukommentieren und wie folgt zu ersetzen:

```
<!-- <jta-data-source>java:/bookingDataSource</jta-data-source> -->
<jta-data-source>java:jboss/datasources/ExampleDS</jta-data-source>
```

Deployen Sie die Applikation, indem Sie die **EAP6\_HOME/standalone/deployments/jboss-seam-booking.ear.failed**-Datei löschen und eine leere **jboss-seam-booking.ear.dodeploy**-Datei in demselben Verzeichnis erstellen.

8. An diesem Punkt deployt die Applikation ohne Fehler, wenn Sie jedoch auf die URL <http://localhost:8080/seam-booking/> in einem Browser zugreifen und "Account Login" versuchen, so erhalten Sie einen Runtime-Fehler "The page isn't redirecting properly". Im nächsten Schritt lernen Sie wie Sie Runtime-Fehler beheben.

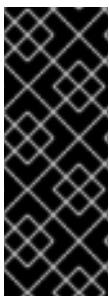
Um mehr zur Fehlerbehebung und der Auflösung von Runtime-Problemen zu erfahren, klicken Sie hier: [Abschnitt 4.3.7, »Fehlerbehebung und Auflösung von Seam 2.2 Booking Archive Runtime Fehlern und Ausnahmen«](#)

Um zum vorherigen Thema zurückzukehren, klicken Sie hier: [Abschnitt 4.3.4, »Migration des Seam 2.2 Buchungsarchivs zur JBoss EAP 6: Schritt-für-Schritt Anleitung«](#)

[Fehler melden](#)

### 4.3.7. Fehlerbehebung und Auflösung von Seam 2.2 Booking Archive Runtime Fehlern und Ausnahmen

Im vorherigen Schritt [Abschnitt 4.3.6, »Fehlerbehebung und Auflösung von Seam 2.2 Booking Archive Deployment Fehlern und Ausnahmen«](#) haben Sie gelernt, wie Deployment Fehler behoben werden. In diesem Schritt beheben wird jeden Runtime Fehler, dem Sie begegnen.



#### WICHTIG

Applikationen, die Hibernate direkt mit Seam 2.2 verwenden, können eine innerhalb der Applikation gepackte Version von Hibernate 3 verwenden. Hibernate 4, welches mittels des org.hibernate Moduls der JBoss EAP 6 bereitgestellt wird, wird von Seam 2.2 nicht unterstützt. Dieses Beispiel soll Ihnen dabei helfen, Ihre Applikation auf der JBoss EAP 6 in Betrieb zu nehmen. Bitte beachten Sie, dass das Packen von Hibernate 3 mit einer Seam 2.2 Applikation keine unterstützte Konfiguration ist.

#### Prozedur 4.11. Fehlerbehebung und Auflösung von Runtime-Fehlern und Ausnahmen

An diesem Punkt sehen Sie beim Deployment der Applikation keine Fehler im Protokoll. Wenn Sie jedoch auf die Applikations-URL zugreifen, so erscheinen Fehler im Protokoll.



1. Problem - javax.naming.NameNotFoundException: Name 'jboss-seam-booking' not found in context "

Wenn Sie auf die URL <http://localhost:8080/seam-booking/> in einem Browser zugreifen, so wird "The page isn't redirecting properly" gemeldet und das Protokoll enthält den folgenden Fehler:

```
SEVERE [org.jboss.seam.jsf.SeamPhaseListener] (http--127.0.0.1-8080-1)
swallowing exception: java.lang.IllegalStateException: Could not
start transaction
    at
org.jboss.seam.jsf.SeamPhaseListener.begin(SeamPhaseListener.java:59
8) [jboss-seam.jar:]
    (... log messages removed ...)
Caused by: org.jboss.seam.InstantiationException: Could not
instantiate Seam component:
org.jboss.seam.transaction.synchronizations
    at org.jboss.seam.Component.newInstance(Component.java:2170)
[jboss-seam.jar:]
    (... log messages removed ...)
Caused by: javax.naming.NameNotFoundException: Name 'jboss-seam-
booking' not found in context ''
    at
org.jboss.as.naming.util.NamingUtils.nameNotFoundException(NamingUti
ls.java:109)
    (... log messages removed ...)
```

#### Das bedeutet:

Eine **NameNotFoundException** deutet ein Problem bei der JNDI-Namensgebung an. Die Regeln für die JNDI-Namensgebung haben sich bei der JBoss EAP 6 geändert, so dass Sie die Lookup-Namen bearbeiten müssen, damit diese den neuen Regeln folgen.

#### So beseitigen Sie das Problem:

Um diesen Fehler zu beheben, sehen Sie im Serverprotokoll nach, welches JNDI-Binding verwendet wurde. Im Serverprotokoll sehen Sie dies:

```
15:01:16,138 INFO
[org.jboss.as.ejb3.deployment.processors.EjbJndiBindingsDeploymentUn
itProcessor] (MSC service thread 1-1) JNDI bindings for session bean
named RegisterAction in deployment unit subdeployment "jboss-seam-
booking.jar" of deployment "jboss-seam-booking.ear" are as follows:
    java:global/jboss-seam-booking/jboss-seam-
booking.jar/RegisterAction!org.jboss.seam.example.booking.Register
    java:app/jboss-seam-
booking.jar/RegisterAction!org.jboss.seam.example.booking.Register
    java:module/RegisterAction!org.jboss.seam.example.booking.Register
    java:global/jboss-seam-booking/jboss-seam-
booking.jar/RegisterAction
    java:app/jboss-seam-booking.jar/RegisterAction
    java:module/RegisterAction
[JNDI bindings continue ...]
```

Insgesamt sind acht INFO JNDI-Bindings im Protokoll aufgeführt, eines für jedes Session-Bean: RegisterAction, BookingListAction, HotelBookingAction, AuthenticatorAction, ChangePasswordAction, HotelSearchingAction, EjbSynchronizations und



TimerServiceDispatcher. Sie müssen die **lib/components.xml**-Datei des WARs so bearbeiten, dass es die neuen JNDI-Bindings verwendet. Beachten Sie bitte, dass im Protokoll die EJB JNDI-Bindings alle mit "java:app/jboss-seam-booking.jar" beginnen. Ersetzen Sie das **core:init**-Element wie folgt:

```
<!--      <core:init jndi-pattern="jboss-seam-booking/#
{ejbName}/local" debug="true" distributable="false"/> -->
<core:init jndi-pattern="java:app/jboss-seam-booking.jar/#{ejbName}"
debug="true" distributable="false"/>
```

Als nächstes müssen Sie die EjbSynchronizations und TimerServiceDispatcher JNDI-Bindings hinzufügen. Fügen Sie der Datei die folgenden Komponentenelemente hinzu:

```
<component class="org.jboss.seam.transaction.EjbSynchronizations"
jndi-name="java:app/jboss-seam/EjbSynchronizations"/>
<component class="org.jboss.seam.async.TimerServiceDispatcher" jndi-
name="java:app/jboss-seam/TimerServiceDispatcher"/>
```

Die components.xml-Datei sollte jetzt wie folgt aussehen:

```
<?xml version="1.0" encoding="UTF-8"?>
<components xmlns="http://jboss.com/products/seam/components"
  xmlns:core="http://jboss.com/products/seam/core"
  xmlns:security="http://jboss.com/products/seam/security"
  xmlns:transaction="http://jboss.com/products/seam/transaction"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
    "http://jboss.com/products/seam/core
http://jboss.com/products/seam/core-2.2.xsd
    http://jboss.com/products/seam/transaction
http://jboss.com/products/seam/transaction-2.2.xsd
    http://jboss.com/products/seam/security
http://jboss.com/products/seam/security-2.2.xsd
    http://jboss.com/products/seam/components
http://jboss.com/products/seam/components-2.2.xsd">

  <!-- <core:init jndi-pattern="jboss-seam-booking/#
{ejbName}/local" debug="true" distributable="false"/> -->
  <core:init jndi-pattern="java:app/jboss-seam-booking.jar/#{
{ejbName}" debug="true" distributable="false"/>
  <core:manager conversation-timeout="120000"
    concurrent-request-timeout="500"
    conversation-id-parameter="cid"/>
  <transaction:ejb-transaction/>
  <security:identity authenticate-method="#
{authenticator.authenticate}"/>
  <component
class="org.jboss.seam.transaction.EjbSynchronizations"
    jndi-name="java:app/jboss-seam/EjbSynchronizations"/>
  <component class="org.jboss.seam.async.TimerServiceDispatcher"
    jndi-name="java:app/jboss-
seam/TimerServiceDispatcher"/>
</components>
```

Deployen Sie die Applikation, indem Sie die **standalone/deployments/jboss-seam-booking.ear.failed**-Datei löschen und eine leere **jboss-seam-booking.ear.dodeploy**-Datei in demselben Verzeichnis erstellen.

2. Problem - Die Applikation deployt und läuft ohne Fehler. Wenn Sie auf die URL <http://localhost:8080/seam-booking/> in einem Browser zugreifen und versuchen sich anzumelden, wird es mit der Meldung "Anmeldung fehlgeschlagen. Transaktion fehlgeschlagen." scheitern. Sie sollten eine Exception Trace im Serverprotokoll sehen:

```
13:36:04,631 WARN [org.jboss.modules] (http-/127.0.0.1:8080-1)
Failed to define class
org.jboss.seam.persistence.HibernateSessionProxy in Module
"deployment.jboss-seam-booking.ear.jboss-seam.jar:main" from Service
Module Loader: java.lang.LinkageError: Failed to link
org/jboss/seam/persistence/HibernateSessionProxy (Module
"deployment.jboss-seam-booking.ear.jboss-seam.jar:main" from Service
Module Loader)
....
Caused by: java.lang.LinkageError: Failed to link
org/jboss/seam/persistence/HibernateSessionProxy (Module
"deployment.jboss-seam-booking.ear.jboss-seam.jar:main" from Service
Module Loader)
...
Caused by: java.lang.NoClassDefFoundError:
org/hibernate/engine/SessionImplementor
  at java.lang.ClassLoader.defineClass1(Native Method)
[rt.jar:1.7.0_45]
...
Caused by: java.lang.ClassNotFoundException:
org.hibernate.engine.SessionImplementor from [Module
"deployment.jboss-seam-booking.ear.jboss-seam.jar:main" from Service
Module Loader]
...
```

#### Das bedeutet:

Die `ClassNotFoundException` weist auf eine fehlende Hibernate Bibliothek hin. In diesem Fall ist es das **hibernate-core.jar**.

#### So beseitigen Sie das Problem:

Kopieren Sie das **hibernate-core.jar** JAR aus dem **EAP5\_HOME/seam/lib/** Verzeichnis ins **jboss-seam-booking.ear/lib** Verzeichnis.

Deployen Sie die Applikation, indem Sie die **standalone/deployments/jboss-seam-booking.ear.failed**-Datei löschen und eine leere **jboss-seam-booking.ear.dodeploy**-Datei in demselben Verzeichnis erstellen.

3. Problem - Die Applikation deployt und läuft ohne Fehler. Wenn Sie auf die URL <http://localhost:8080/seam-booking/> in einem Browser zugreifen, können Sie sich erfolgreich anmelden. Wenn Sie aber versuchen ein Hotel zu buchen, sehen Sie eine Exception Trace.

Um diesen Fehler zu bereinigen müssen Sie zuerst das **jboss-seam-booking.ear/jboss-seam-booking.war/WEB-INF/lib/jboss-seam-debug.jar** entfernen, da es den wahren Fehler verdeckt. An dieser Stelle sollten Sie folgenden Fehler sehen:

```
java.lang.NoClassDefFoundError:
org/hibernate/annotations/common/reflection/ReflectionManager
```

#### Das bedeutet:

Der `NoClassDefFoundError` weist auf eine fehlende Hibernate Bibliothek hin.

#### So beseitigen Sie das Problem:

Kopieren Sie die **hibernate-annotations.jar** und **hibernate-commons-annotations.jar** JARs aus dem **EAP5\_HOME/seam/lib/** Verzeichnis ins **jboss-seam-booking.ear/lib** Verzeichnis.

Deployen Sie die Applikation, indem Sie die **standalone/deployments/jboss-seam-booking.ear.failed**-Datei löschen und eine leere **jboss-seam-booking.ear.dodeploy**-Datei in demselben Verzeichnis erstellen.

4. Runtime- und Applikationsfehler sollten behoben sein.

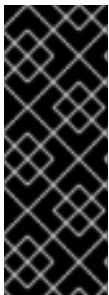
An diesem Punkt deployt und läuft die Applikation ohne Fehler.

Um zum vorherigen Thema zurückzukehren, klicken Sie hier: [Abschnitt 4.3.4, »Migration des Seam 2.2 Buchungsarchivs zur JBoss EAP 6: Schritt-für-Schritt Anleitung«](#)

[Fehler melden](#)

### 4.3.8. Übersicht über eine Zusammenfassung von Änderungen bei der Migration der Seam 2.2 Booking Application

Obwohl es wesentlich effizienter wäre Abhängigkeiten im Voraus zu bestimmen und die impliziten Abhängigkeiten in einem Schritt hinzuzufügen, so zeigt diese Übung, wie Probleme im Protokoll erscheinen und liefert Informationen wie diese behoben werden können. Nachfolgend sehen Sie eine Zusammenfassung der Änderungen an der Applikation bei der Migration zur JBoss EAP 6.



#### WICHTIG

Applikationen, die Hibernate direkt mit Seam 2.2 verwenden, können eine innerhalb der Applikation gepackte Version von Hibernate 3 verwenden. Hibernate 4, welches mittels des `org.hibernate` Moduls der JBoss EAP 6 bereitgestellt wird, wird von Seam 2.2 nicht unterstützt. Dieses Beispiel soll Ihnen dabei helfen, Ihre Applikation auf der JBoss EAP 6 in Betrieb zu nehmen. Bitte beachten Sie, dass das Packen von Hibernate 3 mit einer Seam 2.2 Applikation keine unterstützte Konfiguration ist.

1. Sie haben eine **jboss-deployment-structure.xml**-Datei im **META-INF/**-Verzeichnis des EAR erstellt. Sie haben **<dependencies>** und **<exclusions>** hinzugefügt, um **ClassNotFoundExceptions** aufzulösen. Diese Datei enthält die folgenden Daten:

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-
structure:1.0">
  <deployment>
    <dependencies>
      <module name="javax.faces.api" slot="1.2" export="true"/>
      <module name="com.sun.jsf-impl" slot="1.2" export="true"/>
      <module name="org.apache.commons.logging" export="true"/>
      <module name="org.dom4j" export="true"/>
```

```

        <module name="org.apache.commons.collections"
export="true"/>
    </dependencies>
</deployment>
<sub-deployment name="jboss-seam-booking.war">
    <exclusions>
        <module name="javax.faces.api" slot="main"/>
        <module name="com.sun.jsf-impl" slot="main"/>
    </exclusions>
    <dependencies>
        <module name="javax.faces.api" slot="1.2"/>
        <module name="com.sun.jsf-impl" slot="1.2"/>
    </dependencies>
</sub-deployment>
</jboss-deployment-structure>

```

2. Sie haben die folgenden JARs aus dem **EAP5\_HOME/jboss-eap-5.X/seam/lib/-** Verzeichnis (ersetzen Sie dabei 5.X durch die Version der EAP 5, von der Sie migrieren) in das **jboss-seam-booking.ear/lib/-** Verzeichnis kopiert, um **ClassNotFoundException** aufzulösen:

- o hibernate-core.jar
- o hibernate-validator.jar

3. Sie haben die **jboss-seam-booking.jar/META-INF/persistence.xml**-Datei wie folgt bearbeitet.

1. Sie haben das **jta-data-source**-Element geändert, damit es die mit der JBoss EAP 6 gelieferte Beispieldatenbank verwendet:

```

<!-- <jta-data-source>java:/bookingDatasource</jta-data-source> -
->
<jta-data-source>java:jboss/datasources/ExampleDS</jta-data-
source>

```

2. Sie haben die **hibernate.cache.provider\_class**-Property auskommentiert:

```

<!-- <property name="hibernate.cache.provider_class"
value="org.hibernate.cache.HashtableCacheProvider"/> -->

```

4. Sie haben die **lib/components.xml**-Datei des WAR bearbeitet, damit sie die neuen JNDI-Bindings verwendet

1. Sie haben das bestehende **core:init**-Element wie folgt ersetzt:

```

<!-- <core:init jndi-pattern="jboss-seam-booking/#
{ejbName}/local" debug="true" distributable="false"/> -->
<core:init jndi-pattern="java:app/jboss-seam-booking.jar/#
{ejbName}" debug="true" distributable="false"/>

```

2. Sie haben Komponentenelemente für die "EjbSynchronizations" und "TimerServiceDispatcher" JNDI-Bindings hinzugefügt.

```

<component class="org.jboss.seam.transaction.EjbSynchronizations"

```

```
jndi-name="java:app/jboss-seam/EjbSynchronizations"/>  
<component class="org.jboss.seam.async.TimerServiceDispatcher"  
jndi-name="java:app/jboss-seam/TimerServiceDispatcher"/>
```

[Fehler melden](#)

## ANHANG A. VERSIONSGESCHICHTE

**Version 6.4.0-11.1**  
German Translation

**Mon Nov 09 2015**

**Lisa Stemmler**

**Version 6.4.0-11**

**Tuesday April 14 2015**

**Lucas Costi**

Red Hat JBoss Enterprise Application Platform 6.4.0.GA