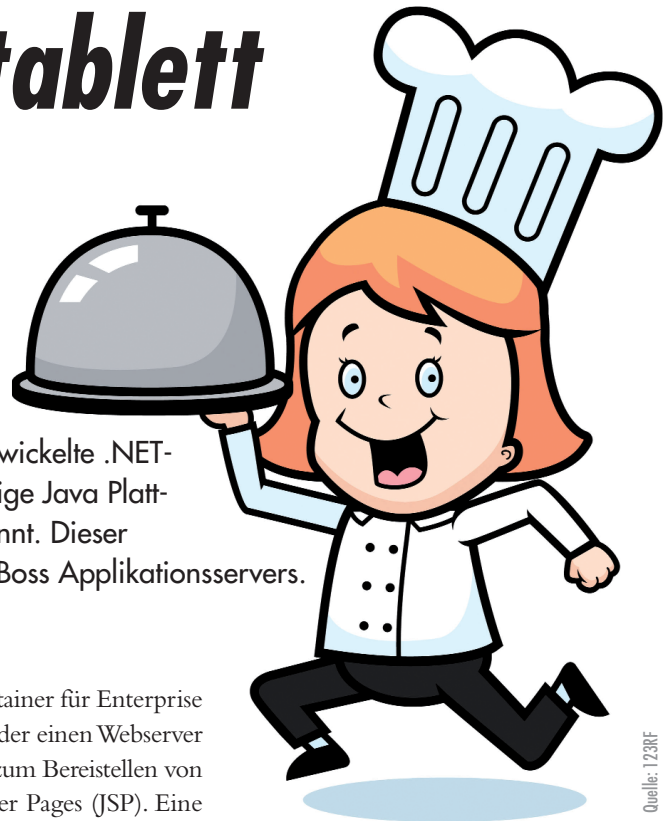




Java Enterprise-Anwendungen mit JBoss bereitstellen Webanwendungen auf dem Silber-tablett

von Thorsten Scherf

Das Entwickeln von Unternehmensanwendungen wird heutzutage immer umfangreicher und komplexer. Steht ein Entwickler vor der Aufgabe eine Unternehmensanwendung zu programmieren, so stehen ihm hierfür im Wesentlichen zwei Software-Architekturen zur Verfügung. Auf der einen Seite existiert die von Microsoft entwickelte .NET-Plattform, auf der anderen Seite die herstellerunabhängige Java Plattform, Enterprise Edition – aktuell auch als Java EE bekannt. Dieser Workshop gibt einen Einstieg in Java EE auf Basis des JBoss Applikationsservers.



Quelle: 123RF

Java EE ist eine Ansammlung diverser Softwarekomponenten und Diensten, die quasi einen De-facto-Standard bei der Entwicklung von Enterprise-Anwendungen darstellen. Durch klar definierte APIs können dabei Komponenten unterschiedlichster Hersteller problemlos miteinander agieren. Jede Java EE basierte Anwendung benötigt natürlich eine entsprechende Umgebung, in der diese ablaufen kann und die die notwendigen Komponenten und Services zur Verfügung stellt, beispielsweise Transaktionsmanagement, Namens- und Verzeichnisdienste, Persistenz- und Deployment-Dienste sowie ein Sicherheits-Framework (siehe Bild 1). Eine vollständige Liste findet sich in der Java EE-Spezifikation [1].

Im kommerziellen Umfeld existieren mit IBM Websphere und BEA WebLogic zwei weitverbreitete Schwergewichte entsprechender Server-Implementierung. Mit JBoss steht solch ein Server als Open Source-Produkt unter der GNU Lesser General Public License (LGPL) zur Verfügung. Dieser stellt alle notwendigen Softwarekomponenten, Dienste und Container der Spezifikation zur Verfügung und erweitert diese um eigene, Server-spezifische, Konfigurationsmöglichkeiten. Beispielsweise enthält

JBoss einen EJB-Container für Enterprise Java Beans (EJB) [2] oder einen Webserver auf Basis von Tomcat zum Bereitstellen von Servlets und JavaServer Pages (JSP). Eine Community-Edition von JBoss, das inzwischen zu Red Hat gehört, steht unter [3] zum Download zur Verfügung.

JBoss installieren

Der JBoss Applikationsserver (AS) steht ebenfalls unter [3] als ZIP-Datei zum Download bereit. Bevor es jedoch an dessen Installation geht, müssen Sie sicherstellen, dass das System entweder über ein aktuelles Java Development Kit (JDK) oder ein Java Runtime Environment (JRE) verfügt. Frühere Versionen von

JBoss AS benötigen zwingend ein JDK, da dieses über einen entsprechenden Compiler für Java-Code verfügt. Dieser ist zum Übersetzen von JSP-Dateien notwendig. Aktuelle Versionen, also AS5 oder AS6, genügt jedoch auch ein JRE, da JBoss eine Eclipse-JDT Bibliothek mit ausgeliefert, die ebenfalls Java-Code übersetzen kann. Handelt es sich bei dem System, auf dem Sie JBoss installieren wollen, jedoch um einen Entwicklungsrechner, so bevorzugen Sie wahrscheinlich den-

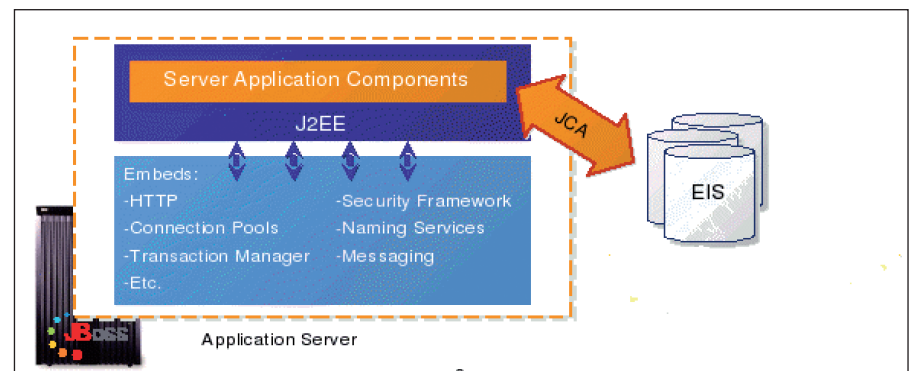


Bild 1: Ein Applikationsserver besteht aus verschiedenen Komponenten, die auf ein Enterprise-Informationssystem (EIS) zurückgreifen können



Die ersten Versionen von JBoss basierten auf einem sogenannten Java-Management-Extension Kernel (JMX). Sämtliche Services des Anwendungsservers wurden hier als Managed Beans (MBeans) in den JMX-Kernel geladen um Ihre Funktionen zur Verfügung zu stellen. Mit der Version AS4 begann dann die Migration auf eine Microcontainer-Architektur, in der nun leichtgewichtige Plain Old Java Objects (POJOs) statt MBeans zum Einsatz kommen. MBeans unterstützt JBoss allerdings immer noch, da der JMX-Kernel nun selbst als POJO auf dem Microcontainer aufsetzt. Für die Zukunft ist geplant, sämtliche MBeans als reine POJOs zu implementieren, so dass der Einsatz des JMX-Kernels nicht mehr notwendig ist

Architektur



noch das JDK. Nach der Installation der Java-Umgebung ist darauf zu achten, dass die JAVA_HOME-Variable auf das Java Root-Verzeichnis zeigt:

```
# export JAVA_HOME=/usr/lib/jvm/
jre-1.6.0-openjdk
# echo $JAVA_HOME
/usr/lib/jvm/jre-1.6.0-openjdk
# java -version
java version "1.6.0_18"
OpenJDK Runtime Environment
(IcedTea6 1.8.3)
(fedora-46.1.8.3.fc13-i386)
OpenJDK Server VM
(build 14.0-b16, mixed mode)
```

Nach diesen Vorarbeiten entpacken Sie das ZIP-Archiv des JBoss AS in ein beliebiges Verzeichnis, beispielsweise nach /opt und passen die JBOSS_HOME-Variable entsprechend an, so dass diese auf das Installationsverzeichnis zeigt. Zur Administration ist ein grundlegendes Verständnis der Verzeichnisstruktur des Servers wichtig. Nach dem Auspacken des Archivs befinden sich im Root-Verzeichnis diverse Ordner:

- * bin-Verzeichnis: Enthält alle Skripte die zum Starten und Stoppen des Servers notwendig sind. Außerdem ist hier auch das JMX-Kommandozeilentool twiddle abgelegt, mit dem Sie den Server administrieren können.
- * client-Verzeichnis: Enthält Bibliotheken die eventuell beim Zugriff von Standalone-Clients benötigt werden. Bei Standalone-Clients handelt es sich beispielsweise um GUI-Clients auf Basis

von Swing oder AWT, Webservice-Clients oder JMS-Clients.

- * doc-Verzeichnis: Enthält hauptsächlich DTD-Dateien und XML-Schemadateien für die JBoss Konfigurationsdateien, da diese auf XML basieren. Konfigurationsbeispiele finden sich unterhalb von docs/examples/.
- * lib-Verzeichnis: Enthält Bibliotheken, die zum Starten des JBoss AS notwendig sind.
- * server-Verzeichnis: JBoss kommt von Haus aus mit verschiedenen Konfigurationen daher. Dieses Verzeichnis enthält diverse Unterverzeichnisse, die jeweils einer bestimmten Konfiguration entsprechen. Beim Starten des Servers können Sie dann bestimmen, welche Konfiguration zum Einsatz kommen soll. Natürlich lassen sich die einzelnen Konfigurationen ändern oder neue hinzufügen. Bild 2 zeigt das Konfigurationsverzeichnis "all". Im Vergleich zur Default-Konfiguration sind hier beispielsweise auch die JBoss-Clusterservices aktiviert.

Durch das Auswählen einer bestimmten Server-Konfiguration können Sie nun also bestimmen, welche JEE-Services und Komponenten der JBoss AS starten soll. Passt keine der vorhandenen Konfigurationen, kopieren Sie einfach einen vorhandenen Ordner, benennen diesen um und passen die Kopie entsprechend an.

Wie das genau geht, erfahren Sie später in diesem Workshop. Möchten Sie den JBoss-Server mit sämtlichen Services starten, so ist hierfür der folgende Aufruf notwendig:

```
# /opt/jboss-5.1.0.GA/bin/run.sh -c
all
```

Auf der Konsole sind nun die Startmeldungen der einzelnen Services zu sehen und am Ende sollte schließlich eine Statuszeile Auskunft darüber erteilen, wie lange die Startup-Prozedur dauerte:

```
19:58:19,965 INFO [ServerImpl]
JBoss (Microcontainer) [5.1.0.GA
(build:SVNTag=JBoss_5_1_0_GA
date=200905221634)] started in
54s:991ms
```

Alternativ ist es auch möglich, den Server im Hintergrund zu starten und die Status-Meldungen der einzelnen Services in eine Log-Datei zu schreiben. Das hat unter anderem den Vorteil, dass die Konsole nicht durch den Server belegt ist:

```
/opt/jboss-5.1.0.GA/bin/run.sh -c
all > console.log &
```

Ein Zugriff auf den Web-Container des Servers sollte zu diesem Zeitpunkt bereits funktionieren (siehe Bild 3). Die Startseite bietet Zugriff auf diverse Online-Res-

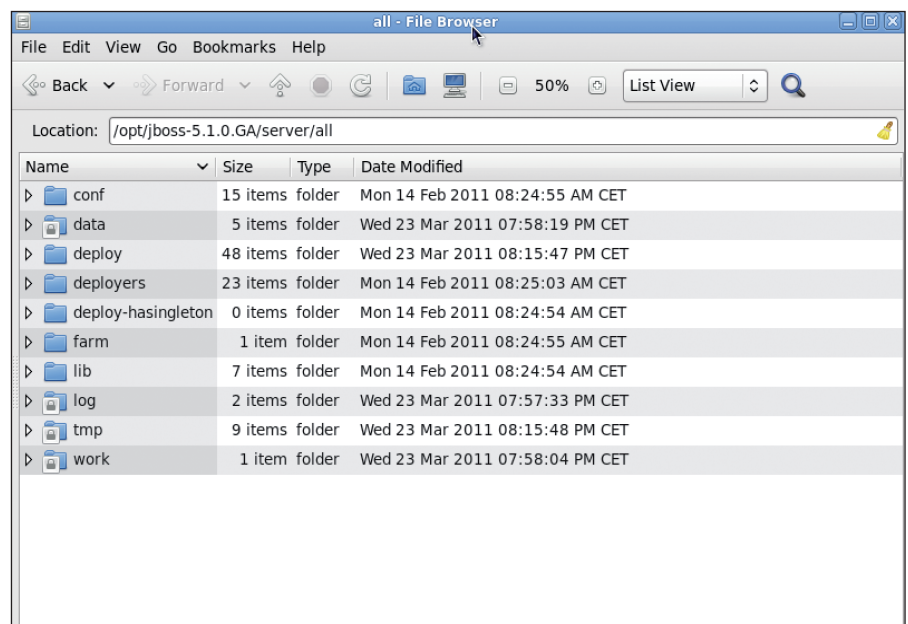


Bild 2: Jedes JBoss-Konfigurationsverzeichnis enthält eine Vielzahl von Unterverzeichnissen, mit denen Sie das Verhalten des Server anpassen können

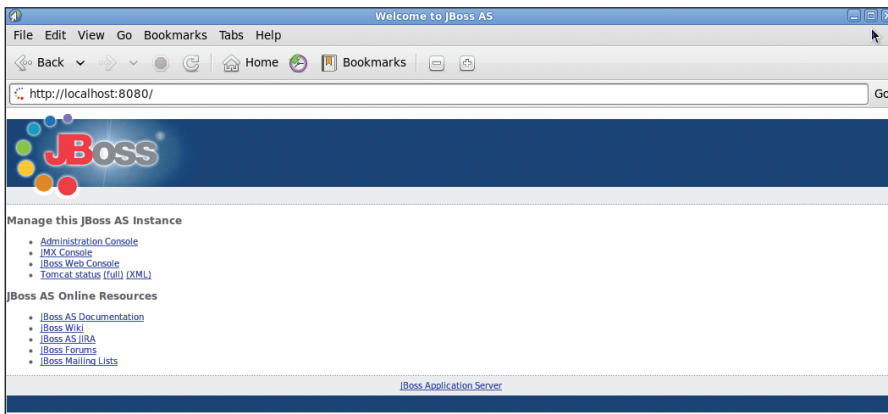


Bild 3: Nach dem Hochfahren des Servers erhalten Sie über die Startseite des Web-Containers Zugriff auf diverse Anwendungen und Online-Ressourcen

sourcen und JBoss interne Applikationen, die sich zur Administration des Servers verwenden lassen.

Erste Java-Webanwendung bereitstellen

Um nun die erste Schritte auf dem neuen Server aufzuzeigen, beginnen wir an dieser Stelle mit dem Bereitstellen einer Java-basierten Webanwendung mit Hilfe des JBoss Web-Services. Dieser basiert auf dem bekannten Tomcat für dynamische Inhalte, kann aber ebenfalls auf die Apache Portable Runtime (APR) zurückgreifen, wenn es darum geht, statische Inhalte schnell auszuliefern. JBoss unterstützt mit dem integrierten Web-Server von Haus aus das sogenannte Hot-Deployment. Hierzu kopieren Sie beispielsweise die gewünschte Webanwendung in ein bestimmtes Verzeichnis der aktiven Server-Konfiguration und JBoss stellt diese Anwendung unmittelbar bereit.

```
<html><head><title>JSP Test</title>
<%!
String message = "Hello, world.";
%>
</head>
<body bgcolor="white">
<h2><%= message%</h2>
<%= new java.util.Date() %>
</body></html>
```

Listing: JSP-Datei

```
<web-app>
<display-name>Hello world</display-name>
</web-app>
```

Listing: Deployment-Deskriptor

Zuerst ist jedoch die entsprechende Anwendung zu erzeugen und vorzubereiten. Da die Entwicklung von komplexen Anwendungen den Rahmen dieses Artikels sprengen würde, greifen wir hier auf die bekannte "Hello World"-Anwendung zurück. Diese liegt hier als JavaServerPage vor. JBoss übersetzt diese dynamisch, erzeugt ein Servlet und stellt dieses einem Client über eine Webschnittstelle zur Verfügung. Es existieren mehrere Methoden diese Anwendung nun bereitzustellen, beispielsweise in Form eines Webarchives (WAR-Archiv). Für dieses Archiv müssen Sie zuerst eine entsprechende Verzeichnis-Struktur erzeugen, wobei die JSP-Datei im Root-Verzeichnis liegt. Hier hin gehört ebenfalls ein Unterverzeichnis WEB-INF mit einem Deployment-Deskriptor namens *web.xml*. Dieser Deployment-Deskriptor ist Teil der JEE-Spezifikation, sodass auch Servlets von anderen Servern unter JBoss lauffähig sind. Server-spezifische Konfigurationsanweisungen liest JBoss aus der Datei *WEB-INF/jboss-web.xml* ein. Dazu zählt beispielsweise die Konfiguration einer Security-Domäne für Ihre Anwendung. Hiermit können dann unter anderem Benutzer authentifiziert und autorisiert werden, bevor diese Zugriff auf die Anwendung, oder Teile davon, erhalten. Die JSP-Datei sowie der Deployment-Deskriptor sind in den beiden Listingkästen dargestellt.

Mit Hilfe des Tools "jar" lässt sich aus diesen Dateien nun ein WAR-Archiv erzeugen (siehe Listing "WAR-Archiv mit jar erzeugen"). Das so erzeugte Archiv kopieren Sie dann in das Host-Deployment-Verzeichnis des aktiven JBoss-Konfigura-

tionsverzeichnis. JBoss nimmt in diesem Falle sofort Notiz von der neuen Web-Anwendung und stellt diese unmittelbar zur Verfügung. Der Aufruf von *http://localhost:8080/helloworld/hello.jsp* im Webbrowser bestätigt dies (siehe Bild 4).

```
# jar -cvf helloworld.war *.jsp
WEB-INF/
added manifest
adding: hello.jsp(in = 159)
(out= 131)(deflated 17%)
adding: WEB-INF/(in = 0)
(out= 0)(stored 0%)
adding: WEB-INF/web.xml(in = 63)
(out= 48)(deflated 23%)
```

```
# cp helloworld.war ../../jboss-
5.1.0.GA/server/all/deploy/
20:15:48,905 INFO [TomcatDeploy-
ment] deploy, ctxPath=/helloworld
```

Statt das WAR-Archiv in das Deployment-Verzeichnis zu kopieren, lässt sich auch das bereits erwähnte Admin-Tool twiddle verwenden. Mit Hilfe der Operation *deploy* der JBoss MBean "jboss.system:service=MainDeployer" stellen Sie die Webanwendung auch dann bereit, wenn Sie beispielsweise gar kein Schreibrecht auf das Deployment-Verzeichnis besitzen. Ein Nachteil dieser Methode besteht jedoch darin, dass die Anwendung nach einem Neustart des Servers nicht wieder automatisch gestartet wird.

```
# ./twiddle.sh invoke "jboss.sys-
tem:service=MainDeployer" deploy
/opt/examples/Helloworld/hello-
world.war
```

Anstatt mit komprimierten Archiv-Dateien, kann JBoss auch mit kompletten Verzeichnis-Strukturen umgehen. Dies ist manchmal einfacher als Archive bereitzustellen, da sich in Verzeichnissen einzelne Dateien leichter verändern oder hinzufügen lassen, ohne das bei jeder Änderung ein neues Archiv zu erzeugen ist. Ein Neustart des Servers ist nach einer Änderung an einer Anwendung nicht notwendig. JBoss erkennt diese automatisch anhand aktualisierter Zeitstempel an den Dateien. Wichtig vor dem Bereitstellen ganzer Verzeichnisse ist jedoch, dass diese den passenden Suffix besitzen, ansonsten weiß

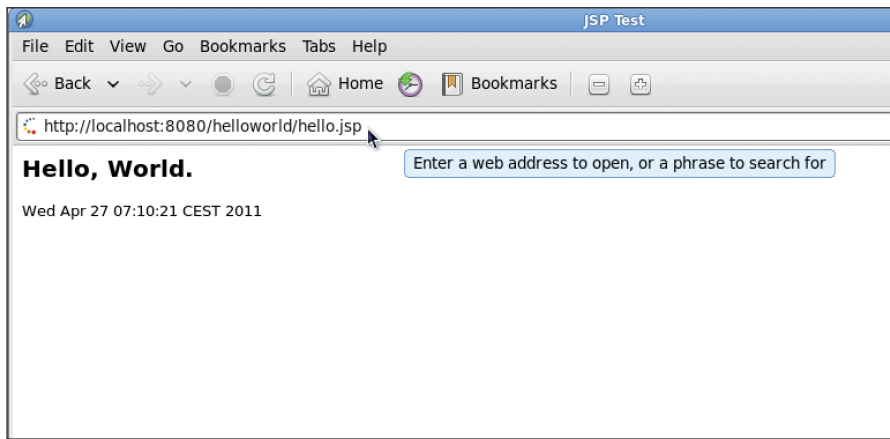


Bild 4: Im Webbrowser lässt sich schliesslich die Ausgabe der Webanwendung bewundern

der Server nicht, welcher Deployer zum Bereitstellen zu verwenden ist:

```
# mv /opt/examples/helloworld/
/opt/examples/helloworld.war/
# rm -f /opt/jboss-
5.1.0.GA/server/all/deploy/
helloworld.war
10:25:22,371 INFO [TomcatDeploy-
ment] undeploy, ctxPath=/
helloworld
# cp -r
/opt/examples/helloworld.war/
/opt/jboss-5.1.0.GA/server/all/
deploy/
10:25:22,412 INFO [TomcatDeploy-
ment] deploy, ctxPath=/helloworld
```

Passende Server-Konfiguration

Nachdem Sie nun wissen, wie Sie eine einfache Webanwendung auf dem JBoss AS bereitstellen können, lohnt es sich, einen genaueren Blick in die Konfiguration des Servers zu werfen. Wie bereits weiter oben erwähnt, enthält JBoss AS mehrere Konfigurations-Verzeichnisse. Beim Starten des Servers wurde in unserem Beispiel die all-Konfiguration ausgewählt. Der entsprechende Konfigurationsordner `/opt/jboss-5.1.0.GA/server/all` enthält diverse Unterverzeichnisse (siehe Bild 5).

Auch hier ist ein grundlegendes Verständnis der einzelnen Verzeichnisse wichtig:

- **conf-Verzeichnis:** Enthält sämtliche Konfigurationsdateien für den JBoss-Kernservice, die einzelnen Services so wie wie für das Logging-Subsystem log4j [4].
- **deploy-Verzeichnis:** Dieser Ordner wurde bereits im letzten Abschnitt vor-

gestellt. Sämtliche Anwendungen, die in dieses Verzeichnis kopiert werden, erkennt JBoss automatisch und stellt diese dynamisch zur Verfügung. Zum Entfernen einer Anwendung ist diese einfach aus dem `deploy`-Verzeichnis zu löschen, oder besser, in ein `undeploy`-Verzeichnis zu verschieben.

- **deployers-Verzeichnis:** Enthält Services zum Erkennen der verschiedenen Anwendungs- und Archivtypen aus dem `deploy`-Verzeichnis.
- **lib-Verzeichnis:** Enthält Bibliotheken die von sämtlichen JEE Services einer gemeinsamen Serverkonfiguration verwendet werden. Spezielle Bibliotheken die nur zu einer einzelnen Anwendung gehören, stellen Sie lieber über die Anwendung selbst zur Verfügung – beispielsweise als Teil eines Webarchives.

Neben diesen Standard-Verzeichnissen erzeugt JBoss noch weitere Ordner zur Laufzeit. Hiervon sind besonders diese interessant:

- **log-Verzeichnis:** Enthält die Dateien `boot.log`, `server.log` und eventuell `audit.log` (je nach Konfiguration). `boot.log` enthält alle Log-Meldung des Start-Prozesses des Servers, bis die Kontrolle an log4j übergeben wird. Dieser Service schreibt die Meldungen dann in die Datei `server.log`. Das Sicherheits-Subsystem von Jboss verwendet die Datei `audit.log` für sicherheitsrelevante Meldungen.
- **work-Verzeichnis:** Hier hält der JBoss Webserver die übersetzten Java-Class Dateien und Servlets auf. Für die oben aufgeführte Webanwendung existieren hier beispielsweise die beiden Dateien

`work/jboss.web/localhost/helloworld/org/apache/jsp/hello_jsp.class` und `work/jboss.web/localhost/helloworld/org/apache/jsp/hello_jsp.java`.

Zum Anpassen des Servers sind sicherlich die Konfigurationsdateien unterhalb von `conf/` am interessantesten. Als Beispiel soll hier das Logging des JBoss-Servers etwas angepasst werden. Wie bereits erwähnt, setzt JBoss das log4j Logging-Framework ein. In der Konfigurationsdatei `conf/jboss-log4j.xml` existieren in der Standardkonfiguration zwei sogenannte Appender. Diese bestimmen wohin die Logmeldungen der JBoss-Services oder der bereitgestellten Anwendungen zu senden sind. Der erste Appender schreibt sämtliche Meldungen aller Loglevel (TRACE, DEBUG, INFO, WARN, ERROR and FATAL) in die Datei `log/server.log` und legt dabei einmal pro Tag eine neue Datei mit einem entsprechenden Datums-Suffix an (siehe Listing "WAR-Archiv mit jar erzeugen"). Soll der Server stattdessen die Log-Datei rotieren, sobald diese ein bestimmte Größe erreicht hat, so sind die Appender-Parameter entsprechend anzupassen (siehe Listing "Anpassen der Appender-Parameter").

Der zweite Appender "CONSOLE" sorgt für eine Ausgabe auf der Konsole. Jedoch findet diese erst ab dem Log-Level INFO statt, wofür der Parameter "`<param name='Threshold' value='INFO'/>`" in der Konfiguration des Appenders zuständig ist. Somit vermeiden Sie, dass das Konsolen-Log mit zu vielen Log-Meldungen

```
<appender name="FILE" class="org.jboss.logging.ap-
pender.DailyRollingFileAppender">
  <errorHandler
class="org.jboss.logging.util.OnlyOnceError
Handler"/>
  <param name="file"
value="${jboss.server.log.dir}/server.log"/>
  <param name="Append" value="true"/>
  <param name="DatePattern" value="'yyyy-
MM-dd"/>
  <layout class="org.apache.log4j.PatternLay
out">
  <param name="ConversionPattern" value="%d
%-5p [%c] (%t) %m%n"/>
  </layout>
</appender>
```

Listing:
WAR-Archiv mit jar erzeugen





überflutet wird. Natürlich ist es auch möglich, für die einzelnen Dateien oder Log-Level einzelne Log-Dateien zu erzeugen. Hierfür existieren in der Kategorie "Limit categories" Einstellmöglichkeiten, die sich auf einem Appender mit einem bestimmten Namen beziehen (dieser ist hier als Referenz anzugeben), und das Logging einer bestimmten Klasse oder einer Anwendung die von diese Klasse abgeleitet ist, ab einen bestimmten Log-Level definiert. Die Konfigurationsdatei bietet hierfür sehr viele Beispiele.

Unnötige Dienste abschalten

In den bisherigen Beispielen wird der JBoss-Server immer mit der Konfigurationseinstellung "all" gestartet. Hierdurch stehen sämtliche Services und Komponenten des Servers zur Verfügung. Für Test- und Entwicklungssysteme mag dies sicherlich in Ordnung sein, für Systeme in der Produktion gilt dies jedoch nicht. Hier gilt der Grundsatz "Weniger ist mehr". Wieso soll der Server Dienste anbieten die gar nicht benötigt werden? Diese stellen nur ein unnötiges Sicherheitsrisiko dar und sollten somit deaktiviert, oder noch besser, entfernt werden.

Mit der Konfiguration "all" startet JBoss 25 verschiedene Services, mit der Konfiguration "default" sind es dagegen lediglich 15 Services. Der Hauptunterschied zwischen den beiden Konfiguration liegt darin, dass mit "all" sämtliche Cluster-Dienste zur Verfügung stehen, mit "default" jedoch nicht. Wenn Sie also

keinen Cluster benötigen, so reicht die Standard-Konfiguration sicherlich aus. Jedoch bietet es sich an, selbst diese Konfiguration noch etwas abzuspecken, falls Sie auf bestimmte Dienste verzichten können. Hierfür ist zuerst ein neues Konfigurationsverzeichnis auf Basis der default-Konfiguration zu erzeugen:

```
# cp -r /opt/jboss-
5.1.0.GA/server/default
/opt/jboss-5.1.0.GA/server/custom
```

In dem Verzeichnis "custom" können Sie nun sämtliche Services deaktivieren, die nicht unbedingt notwendig sind. Beispielsweise soll auf einem Produktionssystem sicherlich nicht die JBoss-Startseite sichtbar sein. Entfernen Sie diese einfach, indem Sie in *custom/deploy/ROOT.war* löschen oder umbenennen. Ist der Mail-Service nicht wirklich notwendig, so löschen Sie die Dateien *custom/deploy/mail-service.xml* und *custom/lib/mail*.jar*. Ein Blick in das *deploy-* und *lib-*Verzeichnis hilft weitere unnötige Services zu identifizieren.

JBoss bietet von Haus aus einige Management-Tools an, dazu zählt beispielsweise die JMX-Console, die unter der URL <http://localhost:8080/jmx-console/> zu erreichen ist. Mit der Konsole haben Nutzer die Möglichkeit, ManagedBeans (MBeans) des Servers zu lesen und zu verändern. Die weiter oben angesprochenen Änderungen am Logging-Subsystem hätten so auch mit der JMX-Console durchgeführt werden können. Somit stellt diese Webanwendung eine mächtige, aber auch gefährliche Möglichkeit dar, den kompletten JBoss-Server zu konfigurieren. Auf einem Produktionssystem ist die Konsole somit entweder zu entfernen (*deploy/jmx-console.war*) oder aber der Zugang ist mit Hilfe einer Benutzerauthentifizierung entsprechend zu sichern.

Authentifizierung und Autorisierung

Das Sicherheitssystem von JBoss basiert auf dem Java Authentication and Authorization Service (JAAS). Über Sicherheits-Domänen legen Sie hier fest, welche Art von Login-Modulen für eine bestimmte Anwendung zum Einsatz kommen soll. Es existieren bereits vorgefertigte Login-

Module für den Zugriff auf Datenbanken, LDAP-Server oder einfache Dateien. Ist eine Webanwendung für eine bestimmte Sicherheits-Domäne konfiguriert, so muss sich ein Benutzer beim Zugriff auf diese authentifizieren und bekommt nach erfolgreicher Authentifizierung und Autorisierung eine bestimmte Rolle zugewiesen. Anhand dieser Rolle kann der Admin festlegen, ob welche Teile der Webanwendung der Benutzer zugreifen darf. Eine vordefinierte Sicherheits-Domäne für die JMX-Console finden Sie im entsprechenden Listing.

Bei diesem Login-Modul handelt es sich um eine einfache Variante, bei der die Authentifizierung und Autorisierung von Benutzern anhand von Dateien stattfindet. Komplexere Beispiele für den Zugriff auf eine Datenbank oder einen LDAP-Server befinden sich jedoch ebenfalls in der Datei *conf/login-config.xml*. Damit eine Anwendung nun weiß, welche Sicherheitsdomäne zum Einsatz kommen soll, ist diese im JBoss-eigenen Deployment-Deskriptor der Anwendung, *jboss-web.xml* zu hinterlegen. Hier tragen Sie den Namen der Domäne ein, mit der diese sich über das Java Naming and Directory Interface (JNDI) auf dem Server angemeldet hat. Im Falle der JMX-Console ist die Änderung also in der Datei *deploy/jmx-console.war/WEB-INF/jboss-web.xml* zu erfolgen:

```
<jboss-web>
  <security-domain>java:/jaas/jmx-
  console</security-domain>
</jboss-web>
```

```
<application-policy name="jmx-console">
  <authentication>
    <login-module
      code="org.jboss.security.auth.spi.UsersRoles
      LoginModule"
      flag="required">
      <module-option
        name="usersProperties">props/jmx-console-
        users.properties</module-option>
      <module-option
        name="rolesProperties">props/jmx-console-
        roles.properties</module-option>
    </login-module>
  </authentication>
</application-policy>
```

```
<appender name="FILE" class="org.jboss.logging.ap-
pender.RollingFileAppender">
  <errorHandler
    class="org.jboss.logging.util.OnlyOnceError
    Handler"/>
  <param name="File"
    value="${jboss.server.log.dir}/server.log"/>
  <param name="Append" value="false"/>
  <param name="MaxFileSize" value="10MB"/>
  <param name="MaxBackupIndex" value="20"/>
  <layout class="org.apache.log4j.PatternLay
  out">
    <param name="ConversionPattern" value="%d
  %p [%c] (%t) %n"/>
  </layout>
</appender>
```

Listing: Anpassen
der Appender-Parameter



Listing: Sicherheits-Domäne
für die JMX-Console





Hiermit ist nun klar, welche Sicherheits-Domäne und somit, welches Login-Modul, für die JMX-Console zum Einsatz kommt. Es fehlt jedoch noch ein Regelwerk, welches bestimmt, wer Zugriff auf welche Ressourcen hat. Diese Information ist der Anwendung nun über den regulären Deployment-Deskriptor `deploy/jmx-console.war/WEB-INF/web.xml` mitzuteilen:

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>HtmlAdap
      <description>Eine beispiel
        hafte Sicherheitseinstellung
        die nur Nutzern mit der
        Rolle JBossAdmin den Zugriff
        auf die HTML JMX-Webappli-
        kation erlaubt
      </description>
    <url-pattern>/*</url-pattern>
    <http-method>GET</http-
      method>
    <http-method>POST</http-
      method>
  </web-resource-collection>
```

```
<auth-constraint>
  <role-name>JBossAdmin</role-
    name>
</auth-constraint>
</security-constraint>
```

Fehlen schließlich noch Benutzeraccount-Daten. Wie im Login-Modul für die Anwendung festgelegt, kommen diese aus den beiden Dateien `props/jmx-console-users.properties` für die Benutzer-Namen und Passwörter, und aus `props/jmx-console-roles.properties` für die Zuordnung einer Rolle zu einem Account. Anhand des Beispiels muss ein Benutzer nun also der Rolle "JBossAdmin" angehören, damit dieser Zugriff auf die Konsole bekommt. Hat alles funktioniert, sollte diesmal beim Aufruf von `http://localhost:8080/jmx-console/` ein Fenster aufgehen, welches den Benutzer zur Authentifizierung auffordert (siehe Bild 5).

Andere Webanwendungen lassen sich nun auf ähnliche Art und Weise für eine Benutzer-Authentifizierung und Autorisierung konfigurieren. Wichtig dabei ist, dass die verwendete Sicherheits-Domäne sich zuvor beim JNDI registriert hat.

Sichere Kommunikation

Abschliessend geht es um die Konfiguration für einen sicheren Zugriff auf eine Webanwendung. Dies ist besonders dann wichtig, wenn die Anwendung sensible Daten überträgt und selbstständig eine Benutzer-Authentifizierung durchführt, wozu dann wahrscheinlich auch die Übertragung eines Benutzer-Passwortes gehört. Für eine gesicherte https-Verbindung ist dem JBoss-Server ein zusätzlicher Web-Connector hinzuzufügen. Vor dessen Konfiguration ist jedoch erst ein Zertifikatsspeicher zu erzeugen. Dieser Speicher enthält ein X.509-Zertifikat welches entweder selbst signiert wurde, oder über die Signatur einer externen Zertifizierungsstelle (CA) verfügt. Mit Hilfe der Java-Anwendung `keytool` lässt sich ein solcher Zertifikatsspeicher anlegen.

Um schließlich den https-Port des Web-connectors zu aktivieren, müssen Sie in der Konfigurationsdatei des JBoss Web-


Services, `deploy/jbossweb.sar/server.xml`, der soeben erzeugte Zertifikatsspeicher, inklusive des dazugehörigen Passwortes, ausführen:

```
<Connector protocol="HTTP/1.1"
  SSLEnabled="true"
  port="8443"
  address="{jboss.bind.address}"
  scheme="https"
  secure="true" clientAuth="false"

  keystoreFile="{jboss.server.home.
    dir}/conf/tuxgeek.keystore"
  keystorePass="redhat"
  sslProtocol = "TLS"
/>
```

Ab nun stehen sämtliche Webanwendungen auch über den gesicherten https-Port 8443 zur Verfügung und bieten somit eine verschlüsselte Übertragung der Daten an.

Fazit

Natürlich kann dieser Artikel nicht sämtliche Konfigurationsmöglichkeiten des JBoss-Applikationsservers vorstellen, dafür ist die Thematik zu umfangreich. Interessante Themen die bisher nicht angesprochen wurden sind beispielsweise die Enterprise Java Beans (EJBs) und das JBoss-Clustering. Mit EJBs ist es möglich, komplexe Anwendungen zu modularisieren und somit die eigentliche Geschäftslogik aus der Anwendung herauszutrennen. Das JBoss-Clustering sorgt für eine hochverfügbare Konfiguration des Servers. Weitere Informationen zu diesen und weiteren Themen bietet die ausführliche Dokumentationsseite des JBoss-Projektes [5]. (dr) 

```
# keytool -genkey -alias example -keystore tuxgeek.keystore -keyalg RSA
keytool -genkey -alias example -keystore tuxgeek.keystore -keyalg RSA
Enter keystore password:
Re-enter new password:
What is your first and last name?
[Unknown]: tiffy.tuxgeek.de
What is the name of your organizational unit?
[Unknown]: GPS
What is the name of your organization?
[Unknown]: Red Hat
What is the name of your City or Locality?
[Unknown]: Stuttgart
What is the name of your State or Province?
[Unknown]: BW
What is the two-letter country code for this unit?
[Unknown]: DE
Is CN=tiffy.tuxgeek.de, OU=GPS, O=Red Hat, L=Stuttgart, ST=BW, C=DE correct?
[no]: yes

Enter key password for <example>
(RETURN if same as keystore password):
Re-enter new password:

# keytool -list -keystore tuxgeek.keystore
Enter keystore password:

Keystore type: JKS
Keystore provider: SUN

Your keystore contains 1 entry

example, Mar 25, 2011, PrivateKeyEntry,
Certificate fingerprint (MD5):
AC:D5:CB:5C:13:9D:EF:F2:85:6B:AB:89:49:F8:4B:3A
```

Listing:
Zertifikatsspeicher mit `keytool`



- [1] Java EE-Spezifikation B7P61
- [2] Java EJB-Technologie B7P62
- [3] JBoss Community Seite B7P63
- [4] log4j Projektseite B7P64
- [5] JBoss Dokumentation B7P65

Link-Codes



EAZ Social Networks

1/1

Anschnitt