## Blitz und Donner

# JBoss AS 7

**Torsten Fink, Moritz Grauel, Heinz Wilming**

*Mit ihrem neuen Applikationsserver in der Version 7 (auch bekannt als AS 7) stellt JBoss/Red Hat eine von Grund auf neu erstellte Architektur des bekannten Java EE-Applikationsservers vor. Sie überzeugt nicht nur durch eine kurze Startzeit und geringen Speicherverbrauch, sondern auch durch ein klares Design mit einem Schwerpunkt auf Modularität. Der AS 7 ist zertifiziert für das Java EE 6 Full Profil und bündelt gereifte und bekannte Infrastrukturkomponenten, wie z. B. HornetQ, Infinispan und Hibernate. Somit können Java EE-konforme Applikationen mit geringen Anpassungen migriert werden. Der Artikel beleuchtet die neue Architektur des AS 7 und dessen konkrete Bedienung aus der Perspektive des Anwendungsentwicklers, die sich im Vergleich zu seinen Vorgängern grundlegend geändert hat.*

## Eine Konterrevolution

Über vier Jahre nach der finalen Version des JBoss 4.0.0 stellte Red Hat im Dezember 2008 endlich den Applikationsserver JBoss 5 vor, basierend auf einer gänzlich neuen Architektur. Doch die neue Version wurde kritisch aufgenommen. Antwortzeitverhalten und Speicherverbrauch konnten nicht überzeugen, wesentlich neue Fähigkeiten blieben aus. Was sich ebenfalls änderte, war die Java Enterprise Edition (Java EE). Im Dezember 2009 wurde die Java EE 6-Spezifikation fertiggestellt. Doch Java EE 6-konforme Applikationsserver ließen zunächst auf sich warten.

Auf Basis der Architektur des JBoss 5 entwickelte Red Hat zunächst den JBoss AS 6, der noch vor Weihnachten 2010 als finale Version erschien. Dieser hinterließ zwar einen durchaus nutzbaren Eindruck, jedoch schien er schon vor seiner Fertigstellung von den JBoss-Entwicklern selbst nur noch stiefmütterlich behandelt zu werden. So wurden alle Folgeversionen der 6er Reihe der Community überlassen und von Red Hat selbst mit nur minimalem Aufwand betreut. Der Fokus der JBoss-Entwickler lag offensichtlich schon auf der nächsten Version.

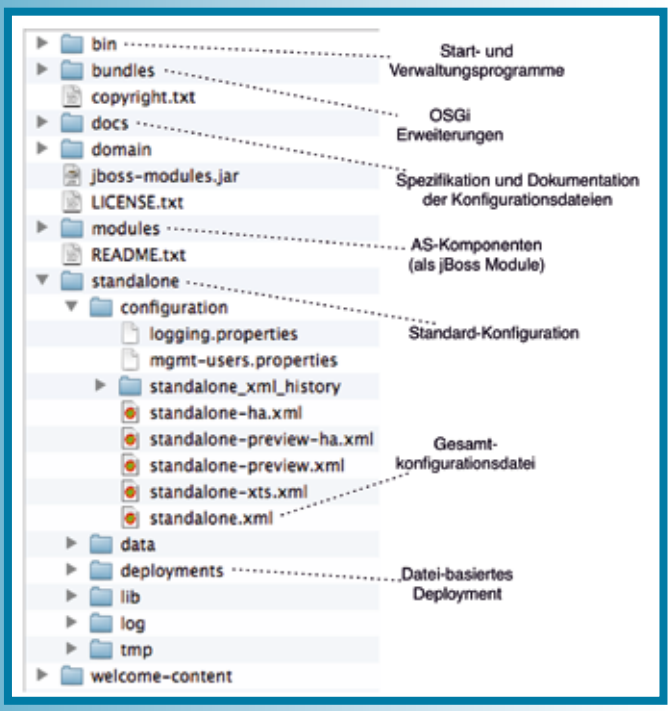


Abb. 1: Verzeichnisstruktur des AS 7

Der offizielle Nachfolger des JBoss AS 6 sollte wieder ein ganz großer Wurf werden. Und so wurde im Juli 2011 mit dem JBoss AS 7 wieder eine komplett neue Architektur vorgestellt, die dieses Mal modernste Höchstleistungen versprach. Vorbei sollten die Zeiten sein, in denen ein Server-Neustart in Minuten gemessen wurde. Schnell wie der Blitz war die Zielvorgabe. Der Codename der Version 7.0.0.Final macht daraus kein Geheimnis: Er lautet selbstsicher Lightning. Auch spätere Versionen blieben in ihren Codenamen bei Blitz und Donner. Die aktuelle 7.1.1.Final etwa hört auf den gleichen Namen wie der griechische Donner-Zyklop: *Brontes*.

Wir wollen in diesem Artikel den JBoss Application Server 7 (AS 7) vorstellen und untersuchen, ob die hohen Ziele, die sich JBoss gesteckt hat, auch erreicht wurden. Dabei gehen wir auf besondere Neuerungen in der Struktur und Architektur des Servers ein und zeigen, wie sich diese in der Praxis auswirken. Da bei der Entwicklung des AS 7 von vornherein auch ein großes Augenmerk auf Testunterstützung gelegt wurde, zeigen wir, wie der AS 7 beim Testen von Anwendung unterstützen kann.

## Erste Berührungen

Der AS 7 hat nicht nur eine neue interne Architektur, sondern auch die Verzeichnisstruktur wurde komplett überarbeitet. Einzig der Ordner „bin" beinhaltet noch die Startup-Skripte. Abbildung 1 zeigt einen groben Überblick über die neue Verzeichnisstruktur. Die Module, welche die technische Basis des AS 7 bilden und später noch genauer beleuchtet werden, liegen im Ordner „modules". Der Ordner „standalone" beinhaltet sowohl die Konfiguration als auch Deployment- und Arbeitsverzeichnisse.

Der AS 7 unterstützt in der Standarddistribution zwei unterschiedliche Konfigurationen: `standalone` und `domain`. Die erste Variante, `standalone`, beinhaltet einen einzelnen, lokal verwalteten Server. Die zweite Variante, `domain`, bietet dagegen ein neues Konzept der zentralen Verwaltung einer ganzen Gruppe von Servern. Diese Gruppe muss nicht zwingend in einem Hochverfügbarkeits-Cluster integriert sein. Beim neuen Domänenkonzept dreht es sich vor allem um eine zentrale Art der Verwaltung. Da sich dieses Konzept noch in der Entwicklung befindet und der Platz in diesem Artikel beschränkt ist, konzentrieren wir uns hier auf die lokal verwaltete Variante.

Im Ordner „docs" bringt der AS 7 seine eigene Dokumentation mit. Der Ordner „welcome-content" beinhaltet statische HTML-Seiten, die beim Webzugriff auf das Wurzelverzeichnis angezeigt werden.

Der Serverstart des AS 7 wird je nach Betriebssystem entweder durch „standalone.sh" oder „standalone.bat" im „bin"-Verzeichnis des AS 7 ausgelöst. Führt man diese Skripte erstmalig aus, begrüßt einen der AS 7 mit ein paar schnell vorbei laufenden Ausgaben, bevor es nach nur wenigen Augenblicken still in der Konsole wird. Wer die Vorgängerversionen kennt, denkt zunächst, hier sei etwas schief gelaufen oder ein besonders langsames Modul lädt vielleicht gerade. Doch bei genauerem Hinsehen lautet die letzte Zeile:

```
...JBoss AS 7.0.2.Final "Arc" started in 1868ms...
```

Und der AS 7 wartet geduldig darauf, dass man etwas mit ihm tut. Einen derart schnellen Start ist man von Applikationsservern nicht gewöhnt.

## Harte Fakten

Um die gefühlten Verbesserungen zu überprüfen, haben wir die Startzeiten und den Speicherverbrauch unterschiedlicher Versionen des JBoss AS gemessen. Zusätzlich haben wir auch aktuelle Tomcat- und Glassfish-Versionen überprüft. Abbildung 2 zeigt die Ergebnisse. Die Messungen wurden auf einem MacBook (2,3 GHz, Intel Core i7) mit den Standardparametern aus der Distribution durchgeführt.

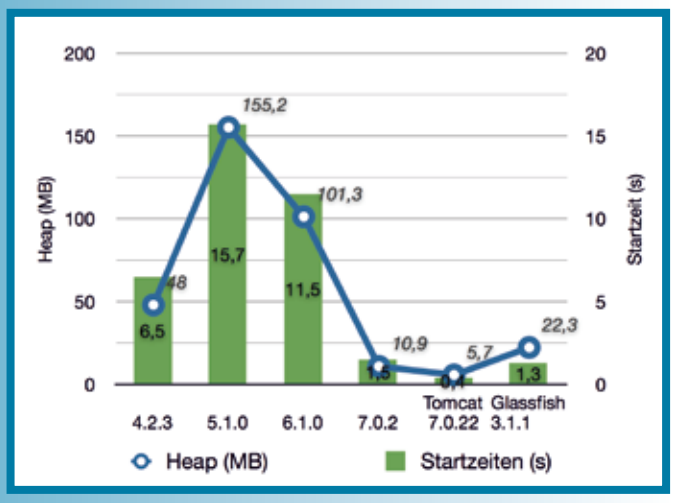


Abb. 2: Vergleich der Ressourcenanforderungen verschiedener Applikationsserver

Beim Wechsel von Version 4 auf 5 sieht man eine deutliche Verschlechterung der Ressourcenanforderung auf fast das Dreifache. Die 6er Version ist zwar im Vergleich zur 5er klar effizienter, aber immer noch doppelt so hungrig wie die alte 4er Version. Die 7er Version ist dagegen grob um den Faktor vier effizienter als der AS 4 bei deutlich vergrößerter Funktionalität.

Die meisten Applikationsserver setzen intern Tomcat als Webserver ein. Die Ressourcenanforderungen von Tomcat stellen damit eine natürliche Grenze dar. Die Messdaten erwecken den Eindruck, als ob ein Drittel der Startzeit beim AS 7 mit dem Hochfahren des Tomcat verbracht wird, der dann etwa die Hälfte des Arbeitsspeichers belegt.

Im Vergleich zu Glassfish (der Java EE-Referenzplattform von Oracle) startet der AS 7 etwa 20 % langsamer, verbraucht aber auch nur die Hälfte an Speicher. Für einen soliden Vergleich der beiden Server müsste man natürlich noch viel mehr in die Details gehen. Unsere Messungen sollen lediglich einen groben Überblick vermitteln. Bezüglich Ressourcenausnutzung und Funktionalität spielen der AS 7 und Glassfish aber inzwischen klar in der gleichen Liga.

## Ein Blick unter die Haube

Mit dem AS 7 hat JBoss den bekannten JBoss AS neu erfunden und sich für einen drastischen Neuanfang entschieden. Der AS 7 basiert jetzt auf *JBoss Modules*, einem neuartigen Modulsystem, welches mit dem althergebrachten Classpath-Mechanismus aufräumt. Es stellt schon heute ein Modulsystem ähnlich dem, wie es JSR 294 irgendwann einmal anbieten wird, zur Verfügung.

Ein Modul im Sinne von JBoss Modules ist eine Sammlung von Klassen und Ressourcen. Ein Modul definiert dabei explizit seine Abhängigkeiten und benötigten Ressourcen sowie optional eine Startklasse. Auf Basis dieser Informationen kann JBoss Modules zur Laufzeit einen Abhängigkeitsgraph erstellen und vor allem analysieren, welche Module voneinander unabhängig sind.

JBoss Modules implementiert auf Grundlage dieser Information einen leistungsfähigen, nebenläufigen Klassenlademechanismus, der in der Lage ist, Klassen und Ressourcen in konstanter Zeit, also O(1), zu laden. Dabei werden nicht, wie bisher, alle vorhanden Klassen in große gemeinsame Klassenrepositories gelegt. Stattdessen werden nur die Klassen verknüpft, die explizit voneinander abhängen. Dies ermöglicht, ähnlich wie OSGi, eine saubere Trennung von Klassen unterschiedlicher Komponenten. Aufgrund des durchgängigen Einsatz von Nebenläufigkeit beim Laden von Klassen verkürzen sich die Startzeiten von Anwendungen auf modernen Mehrkernsystemen deutlich.

Die Module befinden sich im „modules“-Unterordner des JBoss-Home-Verzeichnisses in jeweiligen Unterverzeichnissen entsprechend ihrer *Package*-Namen (also etwa „/modules/org/jboss/weld/api/“). Dort findet sich in der Regel ein „main“-Ordner, der die Moduldefinition und notwendigen Archive (*jars*) enthält. Ein Modul wird jeweils durch eine XML-Datei namens „module.xml“ beschrieben. Es ist somit sehr einfach, den AS 7 um weitere Module zu erweitern. Listing 1 zeigt beispielsweise die Moduldefinition, die nötig ist, um einen PostgreSQL-JDBC-Treiber als Modul bereitzustellen.

```
<module xmlns="urn:jboss:module:1.0" name="org.postgresql">
  <resources>
    <resource-root path="postgresql-9.0-801.jdbc4.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
  </dependencies>
</module>
```

Listing 1: Moduldefinition in „/modules/org/postgresql/main/module.xml“

Die Moduldefinition besteht aus der Angabe der Ressourcen, deren Pfad relativ zur „module.xml“ ist, und den Abhängigkeiten in Form von voll qualifizierten Modulnamen. Selbstverständlich muss sich in diesem Beispiel das „postgresql-9.0-801.jdbc4.jar“ ebenfalls im „main“-Ordner befinden.

## Module, Komponenten und das große Bild

JBoss Modules ist kein vollständiges Komponentensystem, wie etwa OSGi. Es ist lediglich eine dünne Schicht, die einer Anwendung beim Laden zu hoher Modularität verhilft. JBoss Modules ist so konzipiert, dass es mit allen existierenden Bibliotheken funktioniert. Um eine Bibliothek als Modul bereitzustellen, muss lediglich die Modul-Definition erstellt werden.

Auf JBoss Modules setzt der *Modular Service Container* (MSC) auf. MSC bietet ein Komponentensystem, das auf dynamischen Diensten basiert. Es ist interessant zu sehen, dass, obwohl MSC ähnliche Eigenschaften wie OSGi bietet, sich die Entwickler dennoch für eine Eigenentwicklung entschieden haben. Auch bei der Implementierung von MSC wurde durchgehend Wert auf Effizienz und Skalierbarkeit gelegt.

Trotz der neuartigen Architektur kommen viele bekannte Frameworks im AS 7 zum Einsatz. Abbildung 3 zeigt die Gesamtarchitektur sowie einige Komponenten. Bekannte Zeitgenossen sind HornetQ als JMS-fähiger Nachrichtendienst, Weld

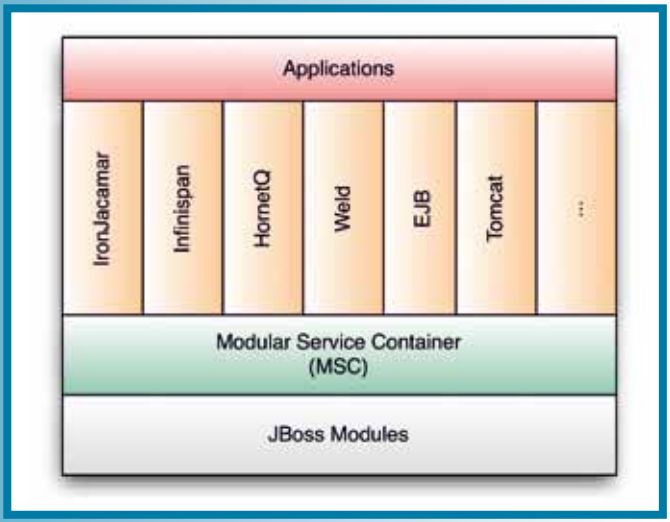


Abb. 3: Grobarchitektur des AS 7 mit seinen Hauptkomponenten

als CDI-Referenzimplementierung, Infinispan als verteilter Cache und Tomcat als Servlet-Container. IronJacamar ist ein neuer JCA-Container, der speziell für den AS 7 entwickelt wurde. Auch der EJB-Container wurde neu entwickelt.

## Konfiguration und Administration

Bei der Konfiguration des AS 7 hat sich im Vergleich zu den Vorgängern einiges getan. Vorbei sind die Zeiten, in denen sich die relevanten Parameter über zahlreiche Dateien verstreuten. Für einen Standalone-Server wird die gesamte Server-Konfiguration an zentraler Stelle vorgenommen. In der Datei „standalone/configuration/standalone.xml" laufen alle Konfigurationen der Subsysteme, wie etwa Logging, Datasources oder Transaktionsmanagement, zusammen. Auch die Schnittstellen, auf denen der AS 7 Ports öffnet, und deren Sicherheitsrichtlinien werden dort verwaltet.

Durch diesen Ansatz ist es leicht, Konfigurationen für verschiedene Umgebungen zu erstellen und miteinander zu vergleichen. An dieser Stelle sei aber auch eine Warnung ausgesprochen. Es empfiehlt sich nicht, diese Datei bei laufendem JBoss von Hand zu editieren, da der AS 7 sie hin und wieder neu schreibt. Stattdessen sollte an laufenden Servern die Konfiguration durch eine der Verwaltungsanwendungen vorgenommen werden, die der AS 7 mitbringt.

Standardmäßig aktiviert der AS 7 eine Web-Konsole unter `http://localhost:9990/console` und eine Kommandozeilenanwendung, durch die sich die Konfigurationsparameter des AS 7 auslesen und setzen lassen. Die Anwendung lässt sich durch das Skript „jboss-cli.sh" starten, setzt jedoch voraus, dass der AS 7 seine native Verwaltungsschnittstelle auf Port 9999 aktiviert hat. Diese Schnittstelle wird ebenfalls in der Datei „standalone.xml" konfiguriert. Bei gestopptem Server spricht nichts dagegen, die Konfiguration auch von Hand direkt im XML vorzunehmen.

Eine weitere wesentliche Änderung hat es beim Installieren von Anwendungen gegeben. In früheren Versionen wurde eine Anwendung einfach in ein Verzeichnis kopiert. Dabei kam es immer wieder vor, dass die Installation gestartet wurde, bevor der Kopiervorgang beendet war. Weitere Probleme ergaben sich insbesondere auf Windows-Systemen durch Dateisperren auf Betriebssystemebene.

Der AS 7 setzt nun auf sogenannte Marker Files, mit denen er den Zustand von Installationen kommuniziert. Über diese Marker lässt sich außerdem auch die Installation steuern. Ein Marker File ist eine leere Datei, die sich im „deployments"-Verzeichnis des Servers befindet und jeweils den gleichen Namen wie das betreffende Archiv hat, erweitert um eine spezifische Endung.

Legt man beispielsweise ein WAR mit dem Namen „my-app.war" im „deployments"-Ordner ab, so legt der AS 7 eine Datei mit dem Namen „my-app.war.isdeploying" an, um zu signalisieren, dass die Anwendung gerade installiert wird. Nach erfolgreicher Installation wird diese entfernt und eine Datei mit dem Namen „my-app.war.deployed" angelegt, um anzuzeigen, dass das WAR erfolgreich installiert wurde. Löscht man diese Datei, so wird „my-app.war" umgehend deinstalliert. Ebenso kann man auch durch das Anlegen einer Datei „my-app.war.dodeploy" eine Neuinstallation erzwingen. Eine Übersicht über alle existierenden Marker Files gibt es unter [MF].

## Testen von Server-Komponenten mit Arquillian

Der JBoss Application Server 7 wurde von Beginn an auf Testbarkeit ausgelegt. Dazu wird das Framework *Arquillian* eingesetzt. Arquillian ist ein Testframework von JBoss, um Integrationstests in einer realen Laufzeitumgebung auszuführen. Das AS 7-Projekt beinhaltet dazu eine Arquillian-Integration, um eine AS 7-Instanz für einen Test bereitzustellen. Die Verwaltung der Laufzeitumgebung, wie zum Beispiel das Starten und Stoppen der Umgebung, kann durch Arquillian erfolgen, indem ein sogenannter Managed Container verwendet wird, oder manuell, indem eine JBoss AS 7-Instanz als Remote Container der Testumgebung bereitgestellt wird. Listing 2 zeigt exemplarisch einen einfachen Arquillian-Test basierend auf JUnit.

```
@RunWith(Arquillian.class)
public class MyBeanTest {
  @Inject
  private MyBean mybean;

  @Deployment
  public static JavaArchive createDeployment() {
    return ShrinkWrap.create(JavaArchive.class, "test.jar")
      .addClass(MyBean.class)
        .addAsManifestResource(EmptyAsset.INSTANCE, "beans.xml");
  }

  @Test
  public void testIsDeployed() {
    assertNotNull(mybean);
  }
}
```

Listing 2: Arquillian-Test basierend auf JUnit

Mittels der Annotation `@Deployment` wird die Methode gekennzeichnet, die das zu installierende Artefakt erzeugt. Das Paketieren der Artefakte erfolgt programmatisch mit dem *ShrinkWrap*-Framework [SW]. In dem Beispieltest wird die Klasse `MyBean` und eine CDI-Markerdatei „beans.xml" zu einem Archiv mit dem Namen „test.jar" gepackt.

In dem Test wird nach Installation des Archivs in den laufenden Container ein Exemplar der CDI-Komponente `MyBean` per Injektion (`@Inject`) bereitgestellt. Der Test prüft nun einfach, ob ein Exemplar auch wirklich erzeugt werden konnte und damit die Installation in dem Container erfolgreich war.

Damit der Arquillian einen AS 7 für einen Test verwalten kann, werden die entsprechenden Bibliotheken benötigt. Lis-

ting 3 zeigt die Konfiguration mit Maven an zwei Beispielen. In dem ersten Maven-Profil **test-jbossas-7-managed** wird ein eingebetteter Applikationsserver verwendet. Das zweite Profil **test-jbossas-7-remote** zeigt die Verwendung eines entfernten Servers. Beide Varianten können mit Arquillian parallel verwendet werden.

```
<profiles>
  <profile>
    <id>test-jbossas-7-managed</id>
    <dependencies>
      <dependency>
        <groupId>org.jboss.as</groupId>
        <artifactId>jboss-as-arquillian-container-managed</artifactId>
        <version>7.1.1.Final</version>
        <scope>test</scope>
      </dependency>
    </dependencies>
  </profile>
  <profile>
    <id>test-jbossas-7-remote</id>
    <dependencies>
      <dependency>
        <groupId>org.jboss.as</groupId>
        <artifactId>jboss-as-arquillian-container-remote</artifactId>
        <version>7.1.1.Final</version>
        <scope>test</scope>
      </dependency>
    </dependencies>
  </profile>
</profiles>
```

Listing 3: Zwei Maven-Profile

Neben der Deklaration der Abhängigkeiten ist es erforderlich, den Pfad zum Applikationsserver bekannt zu geben. Hierzu muss eine „arquillian.xml"-Konfigurationsdatei im Klassenpfad des Tests vorhanden sein. Die JBoss AS 7-Container unterstützen CDI und EJB Injection innerhalb der Testfälle.

Aufgrund der kurzen Startup- und Installationszeiten des AS 7 verkürzt sich die Zeit zum Ausführen von Integrations-Testsuites signifikant gegenüber früheren Versionen und auch der Einsatz einer von Arquillian verwalteten JBoss-Instanz, die nur für die Testausführung gestartet und gestoppt wird, verlängert die Ausführungszeit einer Testsuite kaum.

## Die Praxis

Obwohl das erste finale Release vom JBoss AS 7 (7.0.0.Final) erst im Juli 2011 veröffentlicht wurde, beweist dieser Applikationsserver bereits in vier unserer Kundenprojekte sowie in zahlreichen internen Projekten seinen Reifegrad.

Dabei beglückt er Entwickler und Anwender gleichermaßen. Die kurzen Startup- und Deployment-Zeiten sorgen bei Entwicklern für kurze Turnaround-Zeiten, die sich kaum hinter denen von Frameworks wie etwa Grails verstecken müssen. Dennoch steht dem Entwickler der volle Umfang der Java EE 6-Spezifikation zur Verfügung.

Im Produktivbetrieb macht der deutlich moderatere Speicherverbrauch des AS 7 ihn zu einem sehr umgänglichen Server – sowohl als Windows-Dienst wie auch als Linux-Dämon. Doch auch beim AS 7 sollte man nicht vergessen, die Speicherbegrenzung der JVM (Heap und PermGen) großzügiger zu gestalten. Die Standardeinstellungen der JVM sind heute schlicht nicht mehr zeitgemäß.

Einzig die lange Liste von Fehlerbehebungen, die bisher jedes Pointrelease des AS 7 begleitete, lässt den einen oder anderen konservativen Architekten vielleicht zögern. Diese Fehlerbehebungen sollten jedoch eher als ein gutes Zeichen gewertet werden – zeigt es doch, dass am AS 7 mit voller Energie gearbeitet und verbessert wird!

## Resümee

Im Vergleich zur Vorgängerversion, dem ebenfalls Java EE 6-konformem JBoss AS 6, stellt der AS 7 einen Sprung nach vorne dar und macht Vorfreude auf weitere Versionen und vor allem auf die bald erscheinende Red Hat Enterprise Application Plattform 6. Das JBoss-Team hat mit seiner neuen Architektur des bekannten JBoss eine absolut konkurrenzfähige Plattform vorgestellt, die bereits in ihrem ersten Release überzeugen kann.

## Literatur und Links

**[Aquillian]** http://www.jboss.org/arquillian
**[JavaEE]** Java Specification Requests 316: Java Platform, Enterprise Edition 6 Specification, http://jcp.org/en/jsr/detail?id=316
**[JBoss]** Download der JBoss-Versionen, http://www.jboss.org/jbossas/downloads/
**[JBossAS7]** Homepage des JBoss Application Server 7, http://www.jboss.org/as7
**[JSR294]** Java Specification Request 294: Improved Modularity Support in the Java Programming Language, Early Draft Review, http://jcp.org/en/jsr/detail?id=294
**[MF]** Marker Files, https://docs.jboss.org/author/display/AS7/Application+deployment
**[SW]** Shrinkwrap, http://www.jboss.org/shrinkwrap

**Dr. Torsten Fink** ist Geschäftsführer der Berliner Einheit der akquinet. Neben der Leitung von Java EE-Projekten führt er Architektur- und Technologieberatungen durch.
E-Mail: torsten.fink@akquinet.de

**Moritz Grauel** entwickelt bei der akquinet Java EE-Anwendungen mit Open-Source-Technologien. Sein zweiter Schwerpunkt besteht in der Implementierung von iOS-Applikationen.
E-Mail: moritz.grauel@akquinet.de

**Heinz Wilming** ist seit 2007 bei der akquinet in Berlin als Berater und Entwickler tätig. Als Leiter des JBoss Competence Centers der akquinet liegt seine Spezialisierung in der Entwicklung von Enterprise-Web-Applikationen mit Open-Source-Technologien.
E-Mail: heinz.wilming@akquinet.de