Quarkus macht's vor, WildFly zieht nach

Bernd Müller, Ostfalia

Das aktuelle Zeitgeschehen im Hinblick auf Unternehmensanwendungen im Java-Kontext ist von einer hohen Agilität geprägt. Zu einigen der zuletzt häufiger gehörten Stichwörter oder Namen gehören sicher Jakarta EE, MicroProfile, GraalVM und Quarkus. Um die altehrwürdigen Application-Server scheint es hingegen eher ruhiger zu werden. Wir wollen in diesem Artikel zeigen, dass diese publizistische Ruhe für WildFly zwar auch zutrifft, aber nicht der Realität der Weiterentwicklung entspricht. Beispielhaft wollen wir zwei attraktive Features von Quarkus und deren entsprechende Umsetzung in und mit WildFly vorstellen; das Deployment als JAR sowie ein Entwicklermodus mit Hot Deployment bei Änderungen an Code-Artefakten.

Ein wenig Vorgeschichte

WildFly ist der Nachfolgername des JBoss AS. Nach der Version JBoss AS 7 erschien WildFly 8.0 als nächste Version des Application-Servers. Wie alt der JBoss AS wirklich ist, konnten wir leider nicht in Erfahrung bringen. Die älteste von uns gefundene Version 3.0.0 *[1]* datiert von Mai 2002. Man kann also getrost von über 20 Jahren ausgehen – und in der IT den Application-Server damit mit Fug und Recht ein Urgestein nennen. Nachdem Java EE in der Version 5 viele Altlasten der in Ungnade gefallenen Vorversionen über Bord warf, hielt sich trotzdem standhaft der Mythos der großen und schwerfälligen Application-Server.

Im Juni 2016 stellte JBoss daher WildFly Swarm unter dem Motto „Just enough app-server for microservice-type applications" vor. Die grundlegende Idee ist das Verpacken der tatsächlich genutzten Teile von Java EE mitsamt der Anwendung selbst in ein JAR; quasi „Application-Server to go". Die Anwendung nimmt die verwendeten Teile des Application-Servers mit sich und bildet eine Einheit, in dem Fall ein JAR. Da *Swarm* ein relativ überladener Begriff war, entschied sich JBoss bereits Mitte 2018 zur Umbenennung. Aus *WildFly Swarm* wurde *Thorntail*. Aber auch dies hatte nicht lange Bestand. Nach der großen Aufmerksamkeit, um nicht zu sagen Hype – die Quarkus auf sich gezogen hatte, entschied sich JBoss, Thorntail nicht mehr weiterzuentwickeln. Die Überschneidungen mit Quarkus waren einfach zu groß. Das Angebot von JBoss besteht nun aus Quarkus und WildFly sowie der kommerziellen WildFly-Variante EAP.

WildFly, Galleon und Maven-Plug-ins

Die Weiterentwicklung des WildFly-Application-Servers ist von einer hohen Release-Frequenz gekennzeichnet. Die Version 12 erschien im Februar 2018, die Version 22 im Januar 2021, im Schnitt also alle drei bis vier Monate ein neues Release. Es scheint jedoch so, dass sich dies zukünftig zu Gunsten von Quarkus ändern wird. Zumindest lässt die Analyse der Commits der beiden Projekte auf GitHub diesen Schluss zu. Stuart Douglas, der Hauptentwickler von WildFly, wurde zum Hauptentwickler von Quarkus.

Eine der letzten großen Änderungen innerhalb des WildFly war die interne Modularisierung auf Basis von Galleon-Layers. Galleon *[2]* ist ein Provisionierungswerkzeug, um einen WildFly-Application-Server zu erstellen, der nur die tatsächlich verwendeten Bestandteile von Java EE verwendet. Im Cloud-Zeitalter ist die Minimierung des Application-Servers ein attraktives Ziel. Ein mit Galleon erzeugter WildFly mit JAX-RS und CDI hat eine Größe von 72 MB, der komplette Server eine Größe von 244 MB.

Das altbekannte WildFly-Maven-Plug-in (`org.wildfly.plugins:wildfly-maven-plugin`) kann Anwendungen deployen, redeployen und undeployen. Außerdem kann es den Application-Server herunterladen und starten, um beispielsweise Integrations- oder Systemtests mit Selenium durchzuführen.

Das relativ neue WildFly-Jar-Maven-Plug-in (`org.wildfly.plugins:wildfly-jar-maven-plugin`) erlaubt es, die benötigten Galleon-Layer und die Anwendung in ein JAR zu packen, so wie man das von Thorntail, Quarkus oder auch Spring Boot kennt.

Anwendung und Server verpacken

Der Application-Server besteht nun also aus Galleon-Layern, weiß aber selbst gar nichts von seinem Glück, dass er auch in Teilen distributiert werden kann. Dafür sorgt das WildFly-Jar-Maven-Plug-in. Um einen Eindruck von der Verwendung zu bekommen, zeigt *Listing 1* die Konfiguration des Plug-ins innerhalb des Build-Elements eines POM.

Man erkennt die einzelnen Galleon-Layer in den `<layers>/<layer>`-Elementen: CDI, JSF, EJB, JPA, JAX-RS, BV. Zusätzlich soll die Java EE Default Datasource von WildFly verwendet werden, sodass der H2-JDBC-Treiber sowie die Definition der Datasource ebenfalls zu packen sind. Der Management-Layer wird in der Regel ebenfalls benötigt. Der JAX-RS-Layer hat als Abhängigkeit den Web-Server-Layer, der wiederum den Deployment-Scanner als Abhängigkeit transitiv packt. Soll er wieder entfernt werden, so kann das Element `<excluded-layer>` wie im Beispiel verwendet werden. Wird mit diesem Plug-in gepackt, so entsteht ein ausführbares JAR, das die entsprechenden Galleon-Layer inklusive Abhängigkeiten sowie die Anwendung enthält. Die deployte Anwendung ist unter dem Root-Kontext verfügbar. Alternativ kann ein sogenanntes Hollow-Jar erzeugt werden, bei dem die Galleon-Layer in ein JAR, die Anwendung in ein WAR gepackt werden.

```
<plugin>
  <groupId>org.wildfly.plugins</groupId>
  <artifactId>wildfly-jar-maven-plugin</artifactId>
  <version>${wildfly-jar-maven-plugin.version}</version>
  <configuration>
    <feature-pack-location>...</feature-pack-location>
    <layers>
      <layer>cdi</layer>
      <layer>jsf</layer>
      <layer>ejb</layer>
      <layer>jpa</layer>
      <layer>jaxrs</layer>
      <layer>bean-validation</layer>
      <layer>h2-driver</layer>
      <layer>h2-default-datasource</layer>
      <layer>management</layer>
    </layers>
    <excluded-layers>
      <layer>deployment-scanner</layer>
    </excluded-layers>
  </configuration>
  <executions>
    <execution>
      <goals>
        <goal>package</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

Listing 1

Das zweite interessante Feature des Plug-ins ist die Möglichkeit des automatischen Bauens und Redeployens bei Code-Änderungen, häufig unter *Hot Deployment* oder *Hot Swapping* bekannt. Durch den Aufruf von `mvn wildfly-jar:dev-watch` wird das Plug-in angewiesen, dies durchzuführen. Alle Änderungen unter `src/main/java`, `src/main/webapp` und `src/main/resources` werden erkannt und sind sofort verfügbar, da die Anwendung als sogenanntes *Exploded Deployment* deployt wird. Das bedeutet, dass zum Beispiel Änderungen an Java-Klassen, JSF-Seiten oder Deployment-Deskriptoren erkannt werden und zum Redeployment führen. Das Entwickeln einer Anwendung bekommt neuen Schwung, da der Ändern-Redeploy-Zyklus deutlich schneller ist. Wem der Aufwand des Selbstausprobierens zu hoch ist, findet unter *[3]* ein kurzes Demonstrationsvideo.

Zusammenfassung

Durch die Modularisierung des WildFly-Application-Servers in einzelne Galleon-Layer ist es mit dem WildFly-Jar-Maven-Plug-in möglich, die von einer Anwendung benötigten Implementierungen verschiedener Java-EE-Teilspezifikationen und die Anwendung selbst in ein ausführbares JAR zu packen. Der Bedarf eines eigenständigen Application-Servers entfällt, was vor allem im Container-Umfeld Prozesse vereinfacht und für kleinere Images sorgt. Zusätzlich stellt das Plug-in ein Goal für die Entwicklung bereit, das bei Änderungen verschiedenster Projektdateien ein automatisches Redeployment durchführt. Eine von Entwicklern gern genutzte Vereinfachung.

Referenzen

[1] *https://jbossas.jboss.org/downloads*
[2] *https://docs.wildfly.org/galleon/*
[3] *https://youtu.be/OHXID3XZuOQ*

Bernd Müller
Ostfalia
bernd.mueller@ostfalia.de

Nach seinem Studium der Informatik und der Promotion arbeitete Bernd Müller für die IBM und die HDI Informationssysteme. Er ist Professor, Geschäftsführer, Autor mehrerer Bücher zu den Themen JSF und JPA, sowie Speaker auf nationalen und internationalen Konferenzen. Er engagiert sich im iJUG und speziell in der JUG Ostfalen.