Brauchen wir Java in der Cloud?

Andreas Chatziantoniou, DevTops GmbH und Matthias Fuchs, esentri AG

Seit einiger Zeit mehren sich die kritischen Stimmen: Java hat seine Zeit gehabt; JEE ist zu groß und zu schwer zu verstehen; kann man in der Cloud nicht auch ganz gut ohne Java (JEE) leben? Der Artikel betrachtet diese Entwicklung und geht dabei auf die verschiedenen Aspekte ein, um dann zu einer Antwort darauf zu kommen, ob JEE in der Cloud noch eine Daseinsberechtigung hat.

Java als Sprache kennt einige Facetten, und die Werbe-Aussage von Oracle meint, eine sehr große Anzahl von Geräten würden Java brauchen/ausführen/damit arbeiten. Warum also die Frage, ob Java noch zeitgemäß sei, wenn dies doch so eine Erfolgsgeschichte ist?

Mit der Anzahl der Geräte, die Java unterstützen, und insbesondere mit den Java Cards (Smart Cards auf denen Java läuft), aber gerade auch im Bereich IoT erscheint es so, als ob niemand an der Daseinsberechtigung – oder gerade der Dominanz – von Java im Bereich Mobile/BluRay zweifelt. Die Autoren sind daher davon überzeugt, dass die Diskussion sich nicht an Java als Sprache (in verschiedenen Ausprägungen) orientiert, sondern vielmehr die Frage nach der Zukunft eines Java-basierten Application-Servers gestellt wird. In Bezug auf das JEE-Modell *(siehe Abbildung 1)* können wir diese Kritik verstehen.

Nach vielen Jahren in diversen Projekten, bei denen JEE eingesetzt wurde, lässt sich sagen, dass JEE sehr vollständig ist, aber nur selten alle Komponenten genutzt werden. Nachfolgend eine Übersicht (bei Weitem nicht vollständig) mit Fragen, die sich ein JEE-Application-Server gefallen lassen muss:

- *Präsentation*
  Viele Möglichkeiten, aber nutze ich alle?
- *Business Logic*
  Servlets, JSP, EJBs – sinnvoll, aber alle gleichzeitig?
- *Common Services*
  - JAX-WS, RMI, JTA, JDBC, JMS, JMX, JAAS, JNDI
  - Wahrscheinlich haben 80 Prozent und mehr aller Anwendungen nur JDBC, JMS und EJB
  - JAAS? weblogic/welcome1
- *Backend*
  - Database – ja, gerne
  - Web Services
  - Directory
  - CORBA – in 2017?

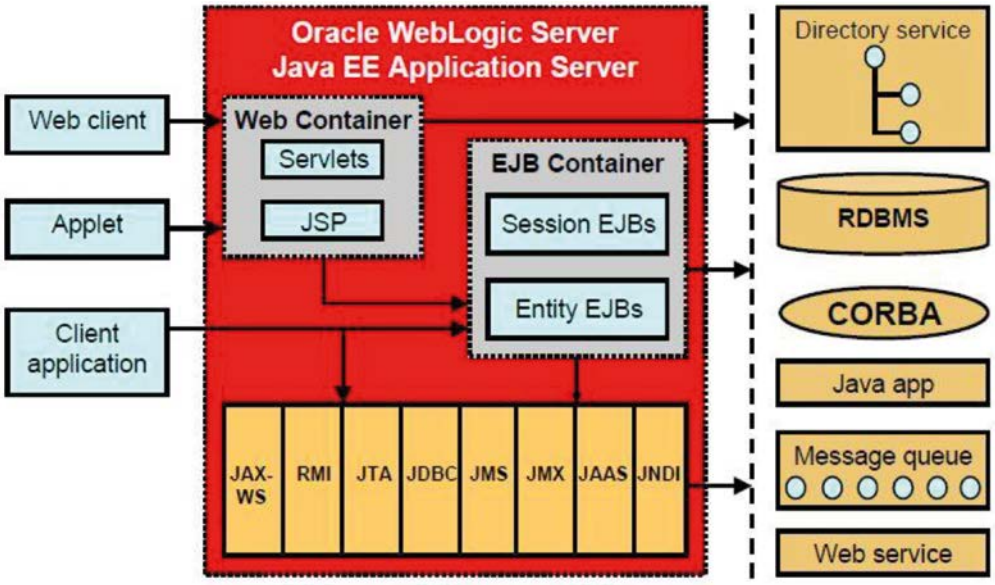

Abbildung 1: Überblick JEE

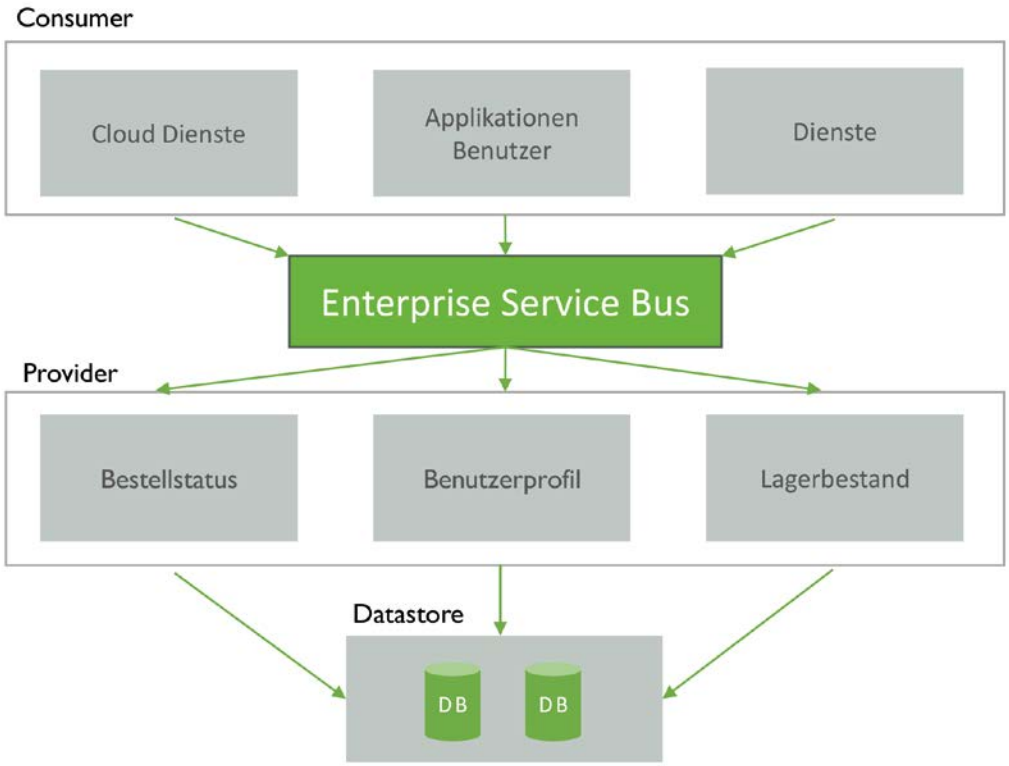

Abbildung 2: SOA-Architektur

Hier wird nach Meinung der Autoren deutlich, aus welcher Richtung die Kritik kommt. Tatsächlich ist die heutige Aufstellung eines Application-Servers darauf fokussiert, alle möglichen Aspekte zu bedienen. Dies hat zur Folge, dass der Umfang des Gebotenen sehr groß ist. Das bedeutet natürlich auch eine dementsprechende Lernkurve. Weiterhin ist der Fußabdruck eines solchen Application-Servers relativ groß und bis alle Komponenten geladen und initialisiert sind, können schon mal ein paar Minuten verstreichen. Wir sollten also auf die Suche gehen, um Lösungen zu finden, die genau den Umfang haben, den ein Projekt braucht, trotzdem skalierbar sind und gegebenenfalls Erweiterungen zulassen, falls sich die Anforderungen ändern.

Von SOA bis Microservice

Serviceorientierung (SOA) war in den letzten Jahren häufig die Basis der Software-Architektur. Es ist eine Sichtweise, wie verteilte Dienste und Services genutzt werden können. SOA-Services werden von unterschiedlichen Bereichen verantwortet und betrieben. Die Dienste sind unabhängig und können über eine Schnittstelle angesprochen werden. Um bei einer höheren Zahl von Diensten die Übersicht zu behalten, werden diese meist über einen zentralen Vermittler verbunden, die Middleware. Hier kommt meist ein Servicebus zum Einsatz, etwa der Oracle-Servicebus. Sowohl der Servicebus als auch die einzelnen Services werden oft auf Application-Servern entwickelt. Datenbanken sind über den Application-Server angesprochen, somit verwenden unterschiedliche Serviceaufrufe oftmals gleiche Datenbanken *(siehe Abbildung 2)*.

Der Microservice-Ansatz geht noch einen Schritt weiter. Ziel ist es, kleine Prozesse zu entwickeln, die unabhängig lauffähig und unabhängig einrichtbar sind. Eine Grundannahme von Microservices ist es, Datenbanken oder andere Storage-Komponenten nicht gemeinsam zu nutzen. Es ergeben sich daher unabhängige Services, die komplett unabhängig entwickelt werden können. Eine zentrale Komponente wie ein Servicebus entfällt *(siehe Abbildung 3)*.

Das Microservice-Design findet in der heutigen Zeit immer mehr Zuspruch. Im Rahmen der Services wird nur ein API bereitgestellt. Die Datenverarbeitung in den Services oder die Entwicklungssprache ist nicht vorgeschrieben. Zusätzlich sind die Services zustandslos, damit es möglich ist, beliebig viele Instanzen einzelner Microservices zu starten. Eine Kommunikation zwischen den Services ist nicht vorgesehen, somit sind Funktionen auf Basis eines Clusters oder eines Remote-Calls, wie sie in Application-Servern vorhanden sind, nicht notwendig.

Java und Microservices

In einem Microservice-Design werden die einzelnen Komponenten völlig unabhängig entwickelt, wie es im Rahmen von agilen Entwicklungsprozessen häufig gefordert wird. Die Kommunikation erfolgt über definierte Schnittstellen (APIs). Eine gemeinsame Basis wie eine gleiche Programmiersprache ist nicht notwendig (Polyglott). Zusätzlich sind die Services meist im Rahmen von Containern bereitgestellt, was problemlos auch unterschiedliche Betriebssysteme möglich macht. Je nach Ziel der Services verwendet man die Sprache oder die Implementation, die eine sinnvolle Umsetzung ermöglicht.

Java bietet sich als Programmiersprache an, ein Wechsel auf Go oder Python ist aber jederzeit möglich, da die Services komplett getrennt ent-

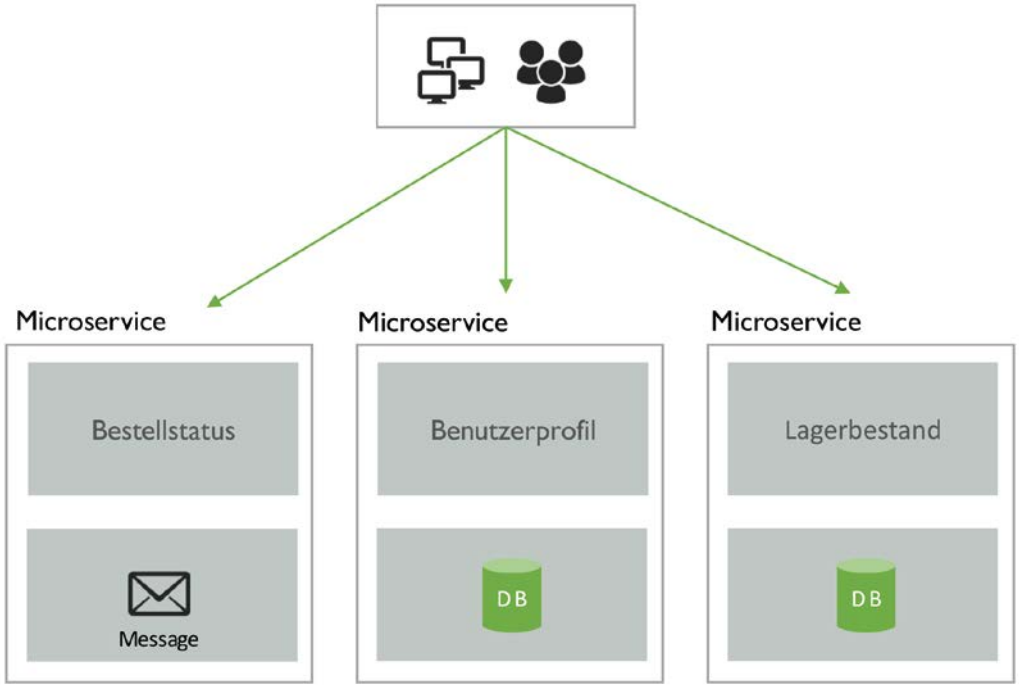

Abbildung 3: Microservice-Architektur

wickelt werden. Ebenso sollte bei Microservices die Datenhaltung getrennt für jeden Container beziehungsweise für jede Komponente erfolgen, um die Unabhängigkeit der Services zu gewährleisten. Dies bietet die Möglichkeit, die optimale Kombination aus Entwicklungsumgebung, Programmiersprache und Speicherpersistierung (Datenbank) zu wählen.

Microservices und Cloud

Es gibt mehrere Gründe, mit Microservices in der Cloud zu starten. Beim Wechsel von SOA-Architekturen auf Microservices ist ein großer Umbau der Infrastruktur notwendig; es ist sinnvoll, dies mit einem Wechsel in eine Cloud zu verbinden. Weiterhin ist das Wesen von Microservices, diese durch Erhöhen oder Verringern der Anzahl der Services (Container) der notwendigen Performance anzupassen. Diese Anpassung der Ressourcen ist in der Cloud sehr einfach möglich, da die Kosten der Infrastruktur teilweise sekundengenau berechnet werden können. Hier sehen wir zwei Richtungen entstehen, die in Zukunft die Einsatzmöglichkeiten von Java beeinflussen werden:

- Container
- Cloud

Im Bereich der Container-Technologie wird sichtbar, dass ein Problem der Application-Server- und Java-Technologie gelöst wird. Oft passiert es, dass auf der Strecke von der Entwicklung über Integration und Abnahme zur Produktion die Umgebungen nicht identisch sind. Dies hat viele Ursachen: Entwickler mit vielen Freiheitsgraden und Geltungsbedürfnissen; Systemadministratoren, die beim Bereitstellen etwas übersehen; manuelle und fehlerhafte Bereitstellungsprozesse; Topologie- und Hardware-Unterschiede.

Beim Einsatz von Containern ist diese Unzulänglichkeit behoben; der Container ist in allen Projektphasen identisch. Hierdurch verschwindet die „Configuration Drift" und damit auch eine Anzahl von Problemen. Nun kommt der Umfang des Application-Servers wieder zum Tragen. Ein vollständiger Application Server wie Oracle WebLogic Server ist im Container erhältlich. Man kann also relativ schnell eine bestehende Anwendung in einen Container packen und in Produktion nehmen.

Da aber im Container bezüglich Skalierung und Hochverfügbarkeit andere Möglichkeiten vorhanden sind, wäre der Übergang zu kleineren, modular aufgebauten Application-Servern denkbar. Kandidaten dafür wären WildFly Swarm, Payara Micro oder TomEE. Diese sind im Container-Umfeld eher anzutreffen als ein klassischer JEE-Full-Stack-Application-Server. Dies hat natürlich auch etwas mit der Lizenzpolitik des Application-Server-Herstellers zu tun. Insbesondere wenn der Application-Server-Anteil am Gesamtsystem eher klein ist, machen sich dessen Kosten schnell bemerkbar. Zusätzlich sind Cluster-Funktionalitäten, wie sie die großen Application-Server-Anbieter im Programm haben, aus Microservice-Sicht nicht notwendig. Application-Server mit geringerem Funktionsumfang bieten somit alle Möglichkeiten, die für eine Microservice-Architektur notwendig sind.

Der zweite Bereich ist die Cloud. Hier ist sowohl für Entwickler als auch für Betreiber ein Umdenken erforderlich. War man bisher vollständig in der Lage, alle Bereiche seiner Infrastruktur zu benutzen, zu konfigurieren, zu untersuchen oder zu integrieren, muss man sich nun mit einem (Web-basierten) User-Interface begnügen. Zugang zu den Logfiles aller Komponenten ist oft nur über Dashboards möglich, der Betrieb wird durch den Cloud-Dienstleister übernommen. Es geht sogar so weit, dass nur der Code beim Anbieter eingespielt wird; welche Server-

Technologie im Hintergrund arbeitet, ist nicht mehr relevant. Im Extremfall spielt man nur noch Funktionen ein (Serverless), somit entfällt sogar die Bindung an die Programmiersprache. Beim Oracle Java Cloud Service ist diese „Encapsulation" am weitesten umgesetzt; man kann demnach auch über Java-as-a-Service sprechen. Der eingebaute JEE-Application-Server (WebLogic) ist nicht mehr direkt sichtbar.

Warum Java und nicht ...

Trotz der Möglichkeit, andere Sprachen einzusetzen, wird für Microservices meist Java verwendet. Dies hat mehrere Gründe. Einerseits gibt es viele Entwickler, die Java beherrschen. Durch einen Wechsel der Sprache müsste man umlernen oder neue Entwickler einstellen. Zudem ist der Produkt-Support mit Java vorhanden und in vielen Projekten erprobt. JEE-Funktionalität geht meist verloren, da es aus Microservice-Infrastruktur- und Architektur-Sicht nicht benutzt wird. Es findet eine Verlagerung von Funktionalität aus dem Application-Server-Umfeld hin zur Infrastruktur statt.

Andere Sprachen wie gerade Python oder Go holen auf. Support ist möglich und es gibt bereits viele Projekte, die erfolgreich damit umgesetzt wurden. In Zukunft werden sich die Sprachen durchsetzen, die für den Use-Case am besten geeignet sind, es ist also eine Abschätzung pro Service notwendig, welche Sprache mit den vorhandenen Ressourcen wie Personal, Support und der geforderten Funktionalität das beste Ergebnis und vor allem eine schnelle und einfache Umsetzung verspricht. Hier ist insbesondere der Aspekt von großen Enterprise Applications zu beachten.

Der Charme von Microservices liegt zurzeit oft darin, leicht zu extrahierende Teile eines Monolithen neu zu bauen. Bieten diese Microservices jedoch den Umfang, den ein (weltweit tätiges) Unternehmen benötigt, und sind Komponenten wie Security oder Data Management darin abgebildet? Wir wissen, dass sich mit Microservices Anwendungen entwickeln lassen, wir wissen aber noch zu wenig davon, ob diese ein vollwertiger Ersatz von JEE-basierten Systemen sind. Mit anderen Worten: Selbst bei kurzen Lebenszyklen von Software (und dem Entwicklungsstatus von anderen Sprachen beziehungsweise Application-Servern) ist Java eine Langzeitgarantie, die für Unternehmen eine große Planungssicherheit bietet.

JEE ist tot ...

Zum heutigen Zeitpunkt (Java 9 wurde gerade veröffentlicht, Java 10 ist in Entwicklung) erscheint den Autoren die Aussage gerechtfertigt, dass Java eine gute Wahl ist, wenn es um den Bau von großen Enterprise Applications geht. Dies ist insbesondere dann wahr, wenn die Erwartung der Laufzeit mehrere Jahre übersteigt. Zudem kann man annehmen, dass Organisationen, die JEE-basierte Application-Server einsetzen und für ihre Anwendungen davon abhängig sind, auch noch mehrere Jahre damit arbeiten werden. In diesem Sinne ist JEE noch lange nicht abgeschrieben.

Da viele JEE-Anwendungen hauptsächlich auf EJB basieren, werden wir in der Zukunft Umbaumaßnahmen sehen, bei denen Spring oder ähnliche leichtgewichtige Frameworks statt EJB zum Einsatz kommen. Dies wird auch Auswirkungen auf den JEE-basierten Application-Server-Markt haben. Die traditionellen Hersteller dieser Application-Server schalten langsam aber sicher auf Angebote um, die entweder in der Cloud laufen (Platform as a Service) und damit die Komplexität zurückdrängen – oder bieten kleinere Application Server an, die in einer modularen Version im Container (generell Docker) lauffähig sind.

Fazit

Die Schlussfolgerung kann eigentlich nur sein: Java is alive and kicking! In den nächsten Jahren werden sowohl Java als auch JEE-basierte Application-Server im Enterprise-Bereich dominant sein und dabei immer öfter durch modernere Entwicklungen ergänzt werden. Konkurrenz durch andere Sprachen braucht Java nicht zu befürchten, Organisationen und Entwickler werden jedoch mehr Auswahl haben, wenn es um die Sprache und/oder den Application-Server geht.

Andreas Chatziantoniou
andreas.chatziantoniou@devtops.gmbh

Andreas Chatziantoniou ist seit dem Jahr 1988 in der IT unterwegs, seit dem Jahr 1998 ausschließlich im Oracle-Technologie-Umfeld. Hier liegt sein Fokus seit dem Jahr 2001 stark auf Application-Servern und der Integration von Systemen. Die Einführung solcher Systeme bei Organisationen und die Übergabe in den Betrieb sind hier immer wiederkehrende Themen. Seit einiger Zeit umfasst sein Arbeitsumfeld auch Container-Technologien und Cloud-Dienste.

Matthias Fuchs
matthias.fuchs@esentri.com

Matthias Fuchs ist seit über 10 Jahren im Umfeld von Oracle-Datenbanken sowie im Bereich Architektur und Infrastruktur von Enterprise-Anwendungen tätig. Er hat seit mehr als 5 Jahren Erfahrungen mit Oracle Engineered Systems wie Oracle Exadata, in Bezug auf Architektur, Tuning und Auditing. Die letzten Jahre erweiterte er sein Beschäftigungsfeld auf NoSQL-Datenbanken und widmete sich IT-Architekturen im Cloud-Umfeld. Momentan arbeitet er in Projekten mit Amazon Webservices oder der Oracle Cloud.