# Brauchen wir Java in der Cloud

**Matthias Fuchs, esentri AG, Nürnberg**
**Andreas Chatziantoniou, Foxglove-IT, Utrecht**

**Schlüsselworte**

Java, Cloud

**Einleitung**

Seit Jahren wird in Projekten mit Java gearbeitet. Wir sollten inzwischen genug Erfahrung besitzen um diese Projekte ohne Probleme über die Bühne zu bekommen. Die Realität zeigt jedoch tagtäglich eine Reihe von Problemen auf.

Die folgenden Fragen werden immer wieder gestellt:

- Warum benutze ich einen riesigen JEE Stack, während ich nur kleine Teile des Standards benutze?
- Jedes Deployment benötigt einen umfangreichen Application Server; ist dieser nicht oversized?
- Gibt es Möglichkeiten, um Software-Entwicklung einfacher zu absolvieren und gleichzeitig komplexe Verfahren anzubieten?

Ansätze wie Spring und Microprofile haben die Java-Applikationsentwicklung verändert. Durch die Verwendung von Container Technologien kommen viele neue Möglichkeiten dazu. Eine Festlegung auf eine Entwicklungssprache ist nicht mehr notwendig. Die Liste der Alternativen ist groß (Javascript, Go, Python ...)

Welche Rolle spielen der Oracle Java Cloud Service, Docker und neue Denkweisen in der Entwicklung wie 12 Factor Applications? Was passiert mit der Fusion Middleware?

**Welche Möglichkeiten habe ich um Java zu benutzen?**

Wenn Du ein Java Entwickler bist und Deine Firma geht in die Cloud, wirst Du Deine Art und Weise ändern müssen wie Du Anwendungen baust und deployst. Dies hängt natürlich auch davon ab, welch Cloud Variante gewählt wird.

Macht es für mich etwas aus ob ich mit IaaS, PaaS oder SaaS arbeiten werde? Wie sieht das aus wenn ich nur noch mit Microservices zu tun habe?

Infrastructure As A Service (IaaS)

Wer IaaS benutzt muss eigentlich den gesamten Stack aufbauen. Dies ist prinzipiell nichts anderes als dies im eigenen Unternehmen zu tun. Hier gibt es dann auch alle bekannten Fehlerquellen. Um Software zu Laufen zu bringen, muss die Umgebung so konfiguriert sein, dass die Software korrekt eingesetzt werden kann. Dies ist eine Aufgabe, die, je nach Komplexität der Umgebung, zwischen ein paar Schritten bis hin zu einigen Hundert Schritten erfordern.

Genau hier (vollständiger Stack: OS, DB, AS, Application) liegt der Punkt der immer öfter zum Reizthema wird. Ist meine Umgebung nicht zu groß? Habe ich nicht zu viele Stellschrauben? Wer blickt eigentlich bei allen Funktionalitäten noch durch?

**Die Frage ist nun ob es nicht sinnvoller ist seine IaaS Umgebung viel kleiner dar zu stellen.**

**Platform as a Service (PaaS)**

PaaS ist hier schon anders, aber bringt mich noch nicht zum Ziel. Alle Komponenten die ich benutze sind mir verborgen – ich sehe nur den Umfang der Funktionalitäten. Diese können ggf. etwas eingeschränkt sein, bieten aber trotzdem mehr als ich will.

**Software as a Service (SaaS)**

Eigentlich ist das Ziel SaaS anzubieten. Auf der Basis der (verborgenen) Plattform entwickle ich eine Anwendung – die GENAU die Funktionalität bietet die mein Endkunde verlangt.

**IaaS für Java Development**

Wenn ich IaaS einsetze wird sich für mich wenig ändern. Meine Entwicklungsaktivitäten werden so bleiben wie sie waren.

**PaaS für Java Development**

Hier wird es für Entwickler eher schwieriger. Die Plattform selber bleibt vor mir verborgen, mein Entwicklungsprozess wird beeinträchtigt und findet eventuell sogar auf einer eigenen Umgebung statt und ich benutze die PaaS Plattform nur für Produktion.

**SaaS für Java Development**

Wenn ich SaaS einsetze bi ich wahrscheinlich am dichtesten beim Ziel. Der Endkunde weiß nicht was es für eine Plattform und Infrastruktur gibt und sieht nur die Anwednung.

Microservices

Obwohl es keine allgemeingültige Definition eines Microservices gibt kann man global sagen es ist eine Sichtweise, die sich an SOA anlehnt. Nur sind meine Services nun kleiner im Angebot und werden entlang ihrer Fähigkeiten eingeteilt, z.B. user interface front-end, recommendation, logistics, billing, etc.

Da wir eine (Seelen)Verwandtschaft mit SOA haben ist die Wahl der Programmiersprache frei, wird aber aller Wahrscheinlichkeit Java sein. Dies kommt nicht von ungefähr, da Microservices oft in Unternehmen eingesetzt werden die schon entsprechende Erfahrungen mit SOA und Java haben.

Entscheidend ist aber der Wandel unter der Motorhaube: während SOA um Web Services zentriert und (teilweise) komplexe XML Strukturen benötigt, kommuniziert der normale Microservice mit REST und hält seine Daten mit JSON.

**Application Servers**

Bevor wir uns die Frage stellen ob Java in der Cloud noch notwendig ist, muss die Frage nach dem Application Server gestellt werden. Was tut dieser Application Server für mich? Vereinfacht kann man sagen ein Application Server stellt ein Gerüst dar in dem eine Reihe von (Java) Standarden umgesetzt sind mit dem Ziel um diverse Funktionalitäten anzubieten. In einer Three-Tier-

Architektur (Drei-Schichten-Modell) wird die Anwendungslogik/Geschäftslogik eines Programms ausgeführt Ein AS vermittelt zu Datenbankservern und erlaubt den Anschluss vieler Clients.

Oder es ist ein Web Application Server (WAS) für Webanwendungen, der für eine Internet- oder Intranet-Anbindung dynamisch HTML-Seiten erzeugt (oder Web Services per SOAP anbietet).

**Warum Java für Applikationsserver**

Vorteile von Java sind immer noch die Unabhängigkeit von einem einzelnen Hersteller. Java wurde von vornherein für moderne Netzwerkanwendungen mit hohen Sicherheitsanforderungen konzipiert.

**Applikationsserver-Erweiterungen**

- Integration mit vorhandenen Legacy- oder ERP-Systemen wird erleichtert
- Der Applikationsserver kann mit allen üblichen Datenbanken verbunden werden, die Informationen mehrerer Datenbanken können verknüpft werden
- Hochverfügbarkeit ist im AS oft sehr einfach möglich

**Was sind die Alternativen für Application Server?**

Während man im AS Umfeld oft eine Umgebung aufgebaut hat in der es eine Anzahl von vorkonfigurierten Komponenten gab und diese so eingesetzt hat, dass z.B. Hochverfügbarkeit möglich war, wird heutzutage oft auf Konzepte wie Docker gesetzt. Hierdurch werden hockskalierbare Umgebungen geschaffen, die im Prinzip nichts anderes sind als eine massive Kopie von einzelnen Umgebungen. Fähigkeiten wie Session Replication und andere werden nicht direkt in Docker unterstützt.

Natürlich gibt es auch andere Application Server, die in ihrem Aufbau kleiner sind. Diese werden dann auch gerade zur Abbildung von Funktionalitäten eingesetzt die in verteilten System notwendig sind (z.B. JMS).

**Wer entscheidet die Strategie?**

Ob wir Java, Spring Boot, Python, Javascript oder NodeJS einsetzen ist nicht nur eine Frage der Technik. Bei der Entscheidung werden viele Faktoren eine Rolle spielen.

Wir denken die folgenden Faktoren spielen hier eine Rolle:

Was ist die Erfahrung der Organisation mit SOA?

Wieviel wurde bisher in SOA / Application Server investiert?

Hat die Organisation Probleme mit der jetzigen Architektur?

Ist eine Erweiterung bzw. Ablösung der Infrastruktur geplant?

Welche Bereiche der Organisation wollen auf eine andere Architektur (microservices statt SOA) umsteigen?

Ist die Software die gekauft wurde auch in einer anderen Architektur lauffähig?

Gerade der letzte Punkt ist wahrscheinlich der wichtigste. Wenn die Hauptanwendung einer Organisation nur auf einer bestimmten Plattform (mit der dazugehörigen Architektur) läuft, dann ist die Motivation um diese umzubauen eher klein.

**Kontaktadresse:**
Matthias Fuchs
esentri AG
Schnieglinger Straße 225b
90427 Nürnberg
Telefon: +49 (0) 172-8288751
E-Mail matthias.fuchs@esentri.com
Internet: www.esentri.com

Andreas Chatziantoniou
Foxglove-IT BV
Texel 18
NL-3524 AP Utrecht
Telefon: +31623259167
E-Mail andreas@foxglove-it.nl