Microservices mit Vert.x und der Oracle SOA Suite

Dominik Galler
esentri AG
76275 Ettlingen

Thomas Schuster
Pforzheim University
75175 Pforzheim

Schlüsselworte

Microservices, SOA, Serviceorientierte Architektur, Vert.x, Oracle SOA Suite

Einleitung

Das Microservice-Architekturmuster ist seit geraumer Zeit eines der Trend-Themen in der IT-Branche. Es haben sich einige Rahmenwerke entwickelt, welche zum Ziel haben, bei der Entwicklung von Microservices zu unterstützen. Eines dieser Rahmenwerke ist Vert.x. Zunächst soll betrachtet werden, welche Charakteristika eine Microservice-Architektur aufweist. Anschließend sollen die Auswirkungen auf Softwaresysteme insgesamt betrachtet werden. Auf dieser Grundlage wird untersucht, wie Vert.x bei der Realisierung von Microservices unterstützt. Danach wird ein kurzer Vergleich zwischen Microservice-Architektur und Serviceorientierter-Architektur gezogen. Dies mündet in einer Analyse, wie Microservices mit der Oracle SOA Suite umgesetzt werden können. Schließlich wird untersucht, wie Vert.x und die Oracle SOA Suite kombiniert werden können, sowie welche Vor- und Nachteile bei dieser Kombination gelten.

Microservice-Architekturen

Die Einführung einer Microservice-Architektur (MSA) hat grundlegende Auswirkungen auf das Gesamtsystem, welche sich zwangsläufig durch die speziellen Eigenschaften der Microservice-Architektur ergeben. Im Folgenden sollen zunächst die wesentlichsten Eigenschaften beschrieben werden. Dabei können die Eigenschaften in die drei Kategorien: Technologie, Architektur und Organisation eingeteilt werden.

Zu den technologischen Eigenschaften gehört zunächst die Verteilung der Dienste im System. Damit einher geht das Prinzip des Shared Nothing (die Services sollen unabhängig voneinander agieren können, das betrifft Logik, Benutzerschnittstelle und Daten sowie ein eigener Prozessraum). Aus diesen beiden Grundsätzen ergibt sich eine weitere Eigenschaft der Microservices, die besonders gute Skalierbarkeit. Weiterhin sind die einzelnen Microservices auf dieser Grundlage absolut unabhängig voneinander deploybar, da sie keine Abhängigkeiten außerhalb ihres eigenen Hoheitsbereichs besitzen.

Daraus ergeben sich die speziellen architektonischen Eigenschaften einer MSA: Durch den eigenen Prozessraum, hat ein Microservice exklusiven Zugriff auf seine Datenmodelle. Anderen Diensten werden diese über API Schnittstellen und geeignete Nachrichtenformate bereitgestellt. Damit hat ein Microservice kaum systemweiten Eigenschaften zu genügen, außer dass er eben diese Schnittstellen (und die korrespondierenden Nachrichtenformate) anbieten und verarbeiten können muss. Aus der Kommunikation über Nachrichten folgt das nur einfache Kommunikationsstrukturen benötigt werden. Die Logik ist im Microservice selbst zu implementieren, die Integrationsschicht stellt somit nur einen einfachen Kanal dar, der zur Übertragung von Nachrichten geeignet sein muss (vgl. Smart Endpoints & Dump Pipes).

Damit ist jede Technologie, die diese Eigenschaften erfüllt, ein potentieller Kandidat um einen Microservice zu entwerfen. Ferner kann eine MSA damit polyglott im Betzug auf Programmiersprachen bzw. zugrundeliegende Technologiestacks aufgebaut werden. Den Entwicklerteams wird damit die Möglichkeit eingeräumt, unabhängig von anderen Teams die am besten passenden Technologien zur Realisierung der Anforderungen an ihren Dienst auszuwählen. Die Anforderung sollen dabei so klein gehalten werden, dass sie eine einzige Geschäftsfunktion umfassen, also dem Single-Responsibility-Prinzip genügen kann.

Die organisatorischen Eigenschaften der MSA betreffen zum einen den Bezug zur Fachlichkeit der kleinen Entwicklerteams. Diese müssen sich somit nicht mit einem komplexen Gesamtprozess beschäftigen, und können sich auf die zu realisierende Fachlichkeit konzentrieren, was zu einer geringeren Belastung der Teammitglieder führen kann. Durch die Unabhängigkeit der Microservices (jeder Dienst liefert eigene GUI, Logik und Datenmodell) ist wesentlich weniger Kommunikations- und Abstimmungsaufwand unter den einzelnen Entwicklerteams notwendig, was eine schnellere Entscheidungsfindung forcieren kann. Letztendlich kann die Qualität der entwickelten Software gesteigert werden, da die Entwicklerteams ihren Microservice auch selbst betreiben (DevOps) sollen. Dies bedeutet, dass sie neben dem Betrieb auch für die Wartung des Dienstes zuständig sind.

Diese Eigenschaften der MSA haben wesentliche Auswirkungen auf die Architektur der Software, das Gesamtsystem und die Organisation. Klassische Softwareentwicklung wurde in einer Art Dreischichten-Architektur geprägt. Dabei gibt es typischer Weise die Frontend-Entwickler, welche die UI-Schicht implementieren. Die Fachkonzeptschicht wird von Middleware-Experten realisiert und die Datenhaltungsschicht von den Datenbankexperten. Alle drei Schichten werden dabei unabhängig voneinander durch die jeweiligen Spezialisten-Teams realisiert. Dieser Architekturstil muss für eine Adaption der MSA aufgebrochen werden, da ein Microservices sowohl seine eigene Benutzeroberfläche, Logik und Datenhaltung besitzt. Das Aufbrechen dieses Architekturstils kann zum Beispiel über vertikale Teilung geschehen. Dabei wird der Geschäftsprozess in einzelne Komponenten – nach Fachlichkeit – aufgeteilt. Für einen Webshop können dadurch beispielsweise die Fachlichkeiten Suche, Produktdaten und Kundendaten resultieren, die entsprechend durch unabhängige Microservices zu realisieren sind. Jeder dieser Services stellt seine eigene Datenschicht, Fachkonzeptschicht und Benutzeroberfläche bereit. Es wird somit allenfalls noch eine Datenintegrationsschicht sowie eine Benutzeroberflächenintegration benötigt. Dies hat, bedingt durch Conways Law, auch Auswirkungen auf die organisatorische Struktur. So generiert nach Conway [1] jede Organisation die ein System entwirft, ein Abbild ihrer organisatorischen Strukturen. Damit muss die Teamzusammensetzung für die Entwicklung eines Microservices dessen Struktur bereits wiederspiegeln. Auf diese Weise geht einher, dass ein Team neben Entwicklern für die Fachkonzeptschicht, auch Entwickler für die Benutzeroberfläche und Datenbankentwickler enthält (Cross-Funktionale Teams). Gegebenenfalls müssen auch Spezialisten für den Betrieb der Microservices mit in das Team integriert werden.

Die Realisierung einer MSA hat damit auch Auswirkungen auf das Gesamtsystem. Zum einen entsteht eine erhöhte Kommunikation durch den Nachrichtenaustausch zwischen den einzelnen Diensten. Dieser Kommunikationsaufwand wird durch die Netzwerkstruktur abgebildet. Damit muss diese dafür geeignet realisiert werden. Weiterhin muss das Gesamtsystem eine Verteilung und Skalierung der Microservices ermöglichen. Die einzelnen Dienste sind vom Gesamtsystem dementsprechend geeignet zu integrieren. Da durch eine hohe Zahl von Diensten eine erhöhte Komplexität in ihrer Betreibung gegeben ist, sollten

geeignete Automatisierungseigenschaften wie beispielsweise Continuous Integration und Delivery eingesetzt werden.

Der Einsatz von Microservices bietet ein hohes Maß an Flexibilität in der Entwicklung und dem Betrieb von Software-Systemen. Durch die kleinen Komponenten, kann eine kürzere Time-To-Market für Erweiterungen und Features erreicht werden, da die einzelnen Teams die Dienste schnell und einfach erweitern oder abändern können sollten. Außerdem kann eine gezielte Skalierung der Microservices (insbesondere vertikal) ressourcenschonend einen Performance-Gewinn erzeugen. So müssen bei dem Beispiel des Webshops, wenn sehr viele Suchanfragen durchgeführt werden nur der Dienst für die Suche skaliert werden, um eine gleichbleibende Qualität des Gesamtsystems zu erreichen. Die kleinen Teams begünstigen zudem agile Entwicklungsmethoden wie Scrum. Dagegen ist zu beachten, dass die reduzierte Komplexität (im Vergleich zu einem Monolithen) in den Microservices auf Netzwerkebene weiter bestehen bleibt. Dies hat eine erhöhte Komplexität dieser zur Folge. Die verteile Datenverarbeitung unterliegt dem CAP-Theorem und auch das Testen und Deployment der Microservices, insbesondere im Hinblick auf Integrationstests kann zu Herausforderungen führen.

Microservices mit Vert.x

Nachdem im vorangegangen Abschnitt betrachtet wurde, welche Eigenschaften und Auswirkungen eine MSA auf das Gesamtsystem hat, soll nun beschrieben werden, wie Vert.x bei der Realisierung von Microservices unterstützen kann. Vert.x ist ein polyglottes, leichtgewichtiges und ereignisorientiertes Rahmenwerk, welches die Realisierung von Microservices begünstigt. Vert.x unterstützt in der aktuellen Version die Programmiersprachen Java, JavaScript, Ceylon, Groovy sowie Ruby. Außerdem gibt es keinen Softwaremonolithen in den die Microservices deployed werden müssen. Es ist unter Java beispielsweise lediglich eine Java Virtuelle Maschine (JVM) notwendig, um Vert.x ausführen zu können. Damit ermöglicht Vert.x eine hohe Flexibilität in der Entwicklung. Die wesentlichen Kernelemente von Vert.x sind neben der Vert.x Instanz selbst die Verticle (diesen repräsentieren dann die Microservices) und der Eventbus.

Ein Verticle beschreibt ein einzelnes Bereitstellungselement. Dieses Bereitstellungselement enthält die Logik und emittiert und konsumiert Events. Diese Events werden über den Eventbus verteilt. Dabei übernimmt das Verticle die Rolle des Smart-Endpoints, da es nicht nur bestimmt, wie die Events emittiert bzw. konsumiert werden, sondern auch welche Folgeaktionen an einen Event anschließen. Der Eventbus ist eine Dump Pipe, er überträgt ledigich die Nachrichten. Dabei werden von Vert.x die Übertragungsmethoden Publish-Subscribe und Point-To-Point unterstützt. Der Eventbus sowie die Verticle werden schließlich in einer Vert.x Instanz gekapselt. Pro JVM wird eine Vert.x Instanz betrieben, diese enthält die Verticle (vgl. Abb. 1). Der Eventbus kann auch über mehrere Hosts verteilt agieren. Dafür kann mit den von Vert.x mitgelieferten Clustermanagern wie bspw. Hazelcast eine Verbindung über mehrere Hosts geschehen.

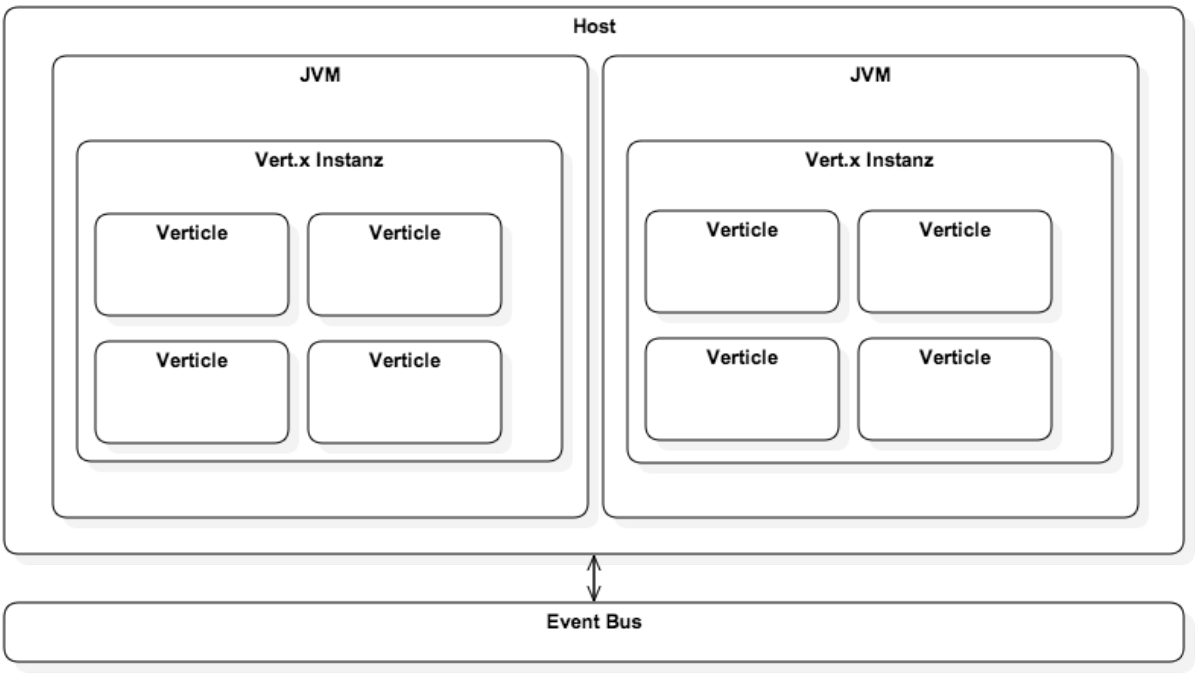

Abb. 1: Struktur des Vert.x Frameworks

Microservices mit der Oracle SOA Suite

Nun soll betrachtet werden, ob und wie Microservices mit die Oracle SOA Suite realisiert und eingesetzt werden können. Dazu ist zunächst festzustellen, in welchem Punkt sich Microservices von Service Orientierter Architektur (SOA) unterscheiden. Bei der Microservice-Architektur im Wesentlichen das Ziel verfolgt wird, eine Anwendung z.B. durch vertikale Teilung in kleine Einzelstücke zu zerlegen, die jeweils nur eine einzige Fachlichkeit abbilden. Somit wird auch eine große Gesamtanwendung in kleine, unabhängige Module zerlegt werden. Automatisch resultiert daraus auch die Wiederverwendbarkeit. Während ähnliche Prinzipien auch bei SOA gelten, steht dort im herkömmlichen Sinne, jedoch die Integration bereits bestehender Altsysteme im Vordergrund. In der betrieblichen Praxis erfordern diese dann oftmals eine komplexe Integrationslogik, um die komplexstrukturierten Geschäftsprozesse zu beherrschen oder zu erweitern. Weiterhin wird dadurch eine Wiederverwendung der Geschäftsprozesse erreicht, während bei MSA die Wiederverwendbarkeit der kleinsten Elemente, also einzelner Microservices das vordergründige Ziel ist.

Das Service-Componente-Architecture-Project (SCA) stellt in der SOA Suite eine zentrale Komponente dar. Es besitzt eine oder mehrere klar definierte Schnittstellen und wird separat auf der SOA Suite ausgeliefert. Intern kann das SCA verschiedene Komponenten zu Realisierung von Geschäftsprozessen und Integrationslogik besitzen. Dazu zählen unter anderem BPEL-Prozesse, Business-Rules und Mediatoren. Eine Skalierung kann durch die Partitionierung im Enterprise Manager der SOA Suite erreicht werden. Eine feinere Steuerung der Skalierung kann durch Zuordnung einzelner Work-Manager-Gruppen zu den entsprechenden Partitionen durchgeführt werden. Wenn man das SCA nun mit einer eigens dafür ausgelieferten UI und einer isolierten Datenbank ausstattet, so kann man mit Hilfe der SOA Suite einen Microservice realisiert. Die wesentlichen Eigenschaften, lose Kopplung und isolierte Entwicklung sind

somit umgesetzt. Durch die Maven-Integration in der SOA Suite 12c kann außerdem eine Automatisierung der Infrastruktur forciert werden. Eine Kommunikation über eine einfache Integrationsschicht (dump pipe) kann weiterhin durch Mediatoren erlangt werden. [2]

SOA Suite und Vert.x

Die Kombination von Vert.x und der SOA Suite ist insofern möglich, als dass ein SCA als klar definierte Schnittstellen auch REST anbieten kann [2]. Durch diese Schnittstellen ist eine Verbindung zu anderen Diensten wie Microservices, z.B. realisiert durch Vert.x, möglich (vgl. Abb. 2).

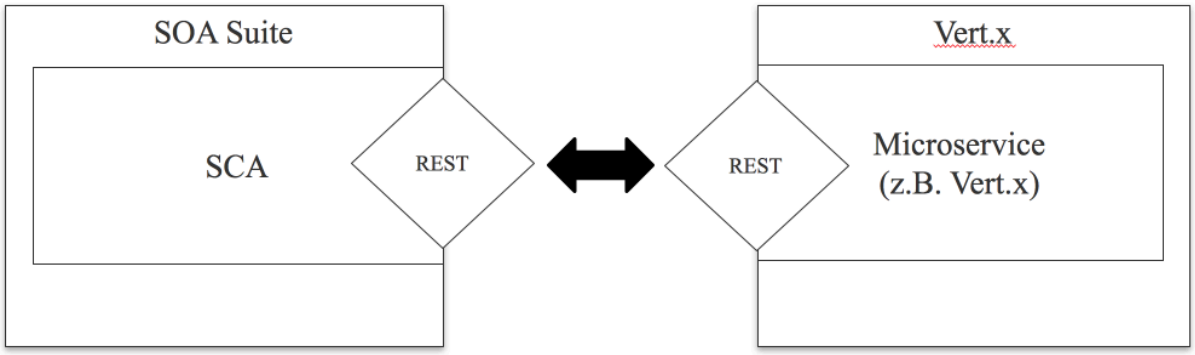

Abb. 2: SCA und Microservice

Diese Kombination kann die Möglichkeit, eine geeignete Lösung zu implementieren, deutlich erweitern. Dann steht der volle Umfang der SOA Suite als auch die Leichtgewichtigkeit von Vert.x an den jeweils am besten geeigneten Stellen zur Verfügung. Man erlangt dadurch Zugriff auf die Vorteile beider Welten. Damit kann an entsprechender Stelle die leichtgewichtige Lösung mit Vert.x realisiert werden, während man an anderer Stelle nicht auf die Vorteile durch die komplexe Integrationslogik der SOA Suite verzichten muss. Außerdem verfügt man über die Monitoring-Eigenschaften, welche die SOA Suite von Haus aus mitbringt um sein System zu überwachen [2]. Dabei muss man aber beachten, dass diese Kombination nicht nur Vorteile aufweist. Durch die Verbindung hat man an vielen Stellen einen doppelten Technologiestack, vgl. Eventbus und Mediator als einfache Integrationsschicht. Weiterhin erhöht man die Komplexität des Systems insbesondere dadurch, dass verschiedene Integrationsschichten innerhalb der einzelnen Technologiestacks zum Einsatz kommen. Insgesamt können sich durch das Zusammenspiel von Vert.x und der Oracle SOA Suite aber neue und interessante Anwendungsgebiete eröffnen.

Quellen

[1] **How Do Committees Invent?, Melvin E. Conway,** Copyright 1968, F. D. Thompson Publications, Inc. Reprinted by permission of *Datamation* magazine, where it appeared April, 1968, URL: http://www.melconway.com/research/committees.html (Abgerufen 23.09.2016)

[2] **Microservices – Architekturmuster oder nur alter Wein in neuen Schläuchen,** Dr. Thomas Schuster und Carsten Wiesbaum, esentri AG, DOAG SOUG Nr. 4, August 2015, ISSN 2364-7191, URL: http://www.esentri.com/wp-content/uploads/2015/09/04-2015-doag-soug-news-web.pdf (Abgerufen 23.09.2016)

Kontaktadresse:

Dominik Galler
esentri AG
Pforzheimer Straße 132
D-76275 Ettlingen

Telefon: +49 (0) 7243 / 354 90 0
Fax: +49 (0) 7243 / 354 90 99
E-Mail dominik.galler@esentri.com
Internet: www.esentri.de

Prof. Dr. Thomas Schuster
Pforzheim University
Tiefenbronner Straße 65
D-75175 Pforzheim

Telefon: +49 7231 28 6111
Fax: +49 7231 28 6110
E-Mail: thomas.schuster@hs-pforzheim.de
Internet: http://www.hs-pforzheim.de