ESB in Rente oder die neue Art, Systeme zu integrieren

Markus Lohn
esentri AG
Ettlingen

Schlüsselworte

ESB, Integration, Microservices, Container, Open Source, OpenShift

Einleitung

Das Verlassen der Komfortzone ist häufig ein schwieriges Unterfangen. Bisher nutzen wir den Oracle Service Bus als Werkzeug für die Entwicklung und Betrieb von Schnittstellen bei einem mittelständischen Kunden. Bis das für uns in diesem Kontext nicht mehr optimal funktionierte. Seit Anfang dieses Jahres implementieren wir Schnittstellen auf Basis einer Microservice-Architektur und Containern. Den Betrieb und die Überwachung dieser Services managen wir durch eine Container-Plattform auf Basis von Red Hat OpenShift. In meinem Vortrag erfahren Sie, warum sich der Kurswechsel mehr als gelohnt hat und wie die neue Technik mehr Geschwindigkeit und Qualität in Implementierung und Betrieb bringt.

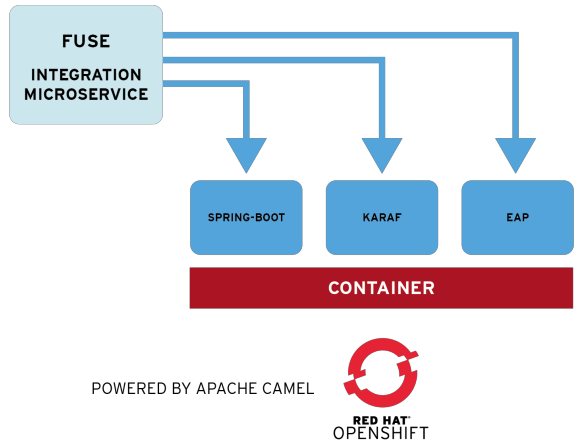

Abbildung 1: Fuse Integration on Spring Boot with OpenShift

Es hat sich gelohnt, weil mehr Entscheidungsfreiheit die Agilität im Projekt erhöhte.

Mehr Entscheidungsfreiheit wird begünstigt durch die Themen Technologie, Minimalismus und Entwicklungsumgebung.

Wir entwickeln und betreiben Schnittstellen in Containern. Jede Schnittstelle wird in einem eigenen Container verwaltet. Container-Technologien, z. B. Docker, sind seit einigen Jahren in der IT etabliert. Innerhalb eines Containers kann jede beliebige Technologie ausgeführt werden. Beliebt ist die Nutzung von Node.js oder Java in Form einer Spring Boot Applikation. Somit besteht die Möglichkeit die passende Technologie für die zu lösende Aufgabe auszuwählen. In unserem Projekt implementierten wir die Schnittstellen auf Basis Red Hat Fuse und Spring Boot. Im Vergleich ist vor allem Java und XML Wissen in einem OSB-Projekt gefragt. Andere Technologien sind nur sehr schwer integrierbar. Der OSB benötigt eine WebLogic Server (WLS) Infrastruktur. WebLogic Server ist ein Java-Server und implementiert Java EE. Somit sind alle Technologien aus dem Java Eco-System grundsätzlich

anwendbar. Eine Migration auf einen anderen OSB oder Java EE Server ist theoretisch möglich, praktisch aber immer eine Neuentwicklung oder mit erheblichen Aufwand verbunden.

WLS stellt eine Vielzahl von Komponenten zur Verfügung, z. B. JMS-Server, HTTP-Server, EJB-Server, MBean-Server, Libraries, Applikationen etc. Viele dieser Komponenten werden aber nicht in jedem Anwendungsfall benötigt, aber dennoch immer geladen. Das benötigt eine lange Startzeit und verschwendet unnötig Ressourcen. Innerhalb eines Containers wird nur das genutzt, was im Dockerfile definiert wurde. Beispielsweise ist für eine Spring Boot Applikation lediglich eine Java Runtime erforderlich.

Mit dem JDeveloper ist die IDE in einem OSB-Projekt fest vorgegeben und kann nicht frei gewählt werden. Die Integration von Apache Maven im JDeveloper mit OSB-Projekten ist nicht optimal gelungen. Es sind doch einige Anpassungen erforderlich, damit ein OSB-Projekt über Maven in eine CI/CD-Pipeline integriert werden kann (-Djava.security.egd=file:///dev/urandom, Projektstruktur standardisieren, clean konfigurieren, Configpläne integrieren). Für die Entwicklung von Services mit Red Hat Fuse kann jede beliebige Java IDE verwendet werden. Die Nutzung eines speziellen Maven Plugins für Fuse ist nicht erforderlich.

Bei der Auswahl der Technologie hat man beim Einsatz von Containern weniger Einschränkungen. Ferner nutzt man wirklich nur die Komponenten, die tatsächlich für die Problemlösung erforderlich sind. Die Entwicklung macht einfach mehr Spaß und die Flexibilität wird erhöht, da kein technologischer Zwang besteht. Durch schnellere Bereitstellung und Feedback kann eine agile Vor-gehensweise viel leichter unterstützt und umgesetzt werden.

Wie schnell können neue Kollegen die Technologie lernen und anwenden? Hat sich das im Projekt gelohnt?

Es hat sich gelohnt, weil neue Kollegen viel schneller integriert wurden.

Zum einen wird das am Zeitaufwand für den Aufbau einer Entwicklungsumgebung deutlich. Die Entwicklungsumgebung für Fuse mit OpenShift aufzubauen benötigt nur zwei Schritte, die in weniger als zwei Stunden durchgeführt werden können. Aufruf eines Installers und Konfiguration der OpenShift Plattform direkt aus der IDE. Im Bereich OSB sind viele Schritte (JDeveloper, Datenbank, WLS und OSB installieren) erforderlich, die u.U. mehrere Stunden Aufwand benötigen. Natürlich existieren Lösungen, um den Prozess und die notwendige Zeit zu reduzieren, z. B. durch Einsatz von Vagrant (https://github.com/markuslohn/vagrant-oracle-soa12213-dev). Jedoch ist der initiale Auf-wand erheblich und an die Zeit für Fuse kommen auch diese Verbesserungen nicht heran.

Zum anderen wird die Einarbeitung durch eine standardisierte Projektstruktur erleichtert. Fuse nutzt Apache Maven als Buildwerkzeug und nutzt die vorgegebene Projektstruktur von Maven. Jeder Entwickler mit Maven-Erfahrung findet sich im Projekt sofort zurecht. In einem OSB-Projekt gibt es einen solchen Standard nicht. Die Projektstruktur muss definiert werden und durch Richtlinien und automatisierte Werkzeuge geprüft und sichergestellt werden.

Ferner unterstützt Fuse durch die Nutzung einer Java DSL vor allem Entwickler beim Einarbeiten und Verständnis der Logik. Im OSB liegen die Sourcen ausschließlich als XML-Dateien vor. Ohne die Nutzung vom JDeveloper, der die XML-Dateien in einer grafischen Anzeige aufbereitet, wäre eine Einarbeitung nicht denkbar.

In unserem Projekt integrierten wir sowohl junge Kollegen mit wenig Projekterfahrung also auch erfahrene Senioren. Insgesamt betrachtet, benötigten die neuen Kollegen ca. einen Tag für ein Onboarding im Projekt. Bereits am zweiten Tag wurden User Stories aus dem Backlog erfolgreich bearbeitet. In OSB-Projekten konnten wir das bis dato so nicht erleben.

Durch das schnellere Aufsetzen der Entwicklungsumgebung und den standardisierten Aufbau von Fuse-Projekten konnten Entwickler viel schneller im Projekt produktiv werden. Die Einarbeitung in die Technologie wird durch die große, engagierte Community und die vielen Ressourcen im Netz erheblich unterstützt und verkürzt. Als Ergebnis konnten sich die Entwickler schneller im Projekt integrieren und auf die Lösung von Fachproblemen fokussieren.

Wie aber war die allgemeine Entwicklungs-Experience? Wurde erhoffte Akzeptanz erreicht und lohnte sich das wirklich?

Es hat sich gelohnt, weil die Entwicklung neuer Integrationen "leichter von der Hand ging".

Wiederverwendung konnte in Fuse mit bekannten Mitteln, z. B. Vererbung, Auslagerung in Libraries oder Copy & Paste, sehr einfach erreicht werden. Im OSB funktioniert das zwar auch, aber über die Anlage eines separaten OSB-Projektes. In diesem neuen OSB-Projekt können das z. B. XSDs, WSDLs, XLSTs oder Alerts zusammengefasst werden. Das Referenzieren dieser Artefakte muss dann jedoch über relative Pfade sichergestellt werden. Das hatte natürlich Auswirkungen auf Entwicklung und Betrieb.

Refactoring kann mit Fuse einfach in der genutzten IDE ausgeführt werden. Ob Packages neu organisieren oder Umbenennungen von Klassen stellt kein Problem dar und wird optimal durch die Entwicklungsumgebung unterstützt. Ab der Version 12c wurde das Refactoring im OSB erheblich verbessert, vor allem in Bezug auf Namespaces. Aber dennoch kann es vereinzelt heute noch notwendig werden, Dinge manuell in XML-Dateien zu ändern.

Eine Fuse Spring Boot Applikation kann viel schneller getestet werden als beim OSB. Eine Spring Applikation kann auf der Entwicklungsumgebung in weniger als 30 Sekunden gestartet werden. Das ist mit dem OSB nie erreichbar. Ferner hatten wir immer wieder Probleme mit git und dem Merge von XML-Dateien im OSB. Bei Fuse mit Java hatten wir diese Probleme nicht mehr feststellen können.

Die Unterstützung von Patterns ist bei Fuse fester Bestandteil im Framework. Im Bereich Integration existiert ein Standardwerk von Gregor Hope: "Enterprise Integration Patterns". Für viele der beschriebenen Patterns gibt es bei Fuse eine eigene Komponente, z. B. Routing Slip, Content Enricher. Im Bereich OSB gibt es keine speziellen Komponenten die Patterns implementieren. Einige Integrationspatterns werden im Produkt direkt unterstützt. Andere Patterns müssen manuell nachprogrammiert werden, z. B. Content Enricher oder Wire Tap.

Durch die Anwendung von Integration- und Software Patterns sowie die sehr gute Werkzeugunterstützung wurde die tägliche Arbeit jedes Entwicklers unterstützt und erleichtert. Generell war die Akzeptanz und Zufriedenheit bei den Projektmitgliedern insgesamt höher als zuvor.

Hatte das Vorgehen auch Auswirkungen auf die Qualität? Lohnte sich der Umstieg?

Es hat sich gelohnt, weil wir die Qualität enorm gesteigert haben.

Vorgaben hinsichtlich der Gestaltung von Sourcecode oder Architektur konnten wir durch Einsatz entsprechender Werkzeuge, automatisiert für alle Schnittstellen mit Fuse, prüfen und sicherstellen. Die Prüfungen wurden auch in den Buildprozess integriert. Für OSB-Projekte existiert hier keine

Werkzeugunterstützung. Somit sind solche Vorgaben zunächst einmal nur manuell durchführbar und damit ist der gesamte Prozess fehleranfällig.

Der große Vorteil bei Fuse besteht darin, jede einzelne Komponente durch einen Unit Tests prüfen zu können. Das Mocking von Endpunkten ist bereits im Framework enthalten und somit entfällt die Nutzung eines zusätzlichen Mocking-Werkzeuges. Somit kann eine Schnittstelle ausgiebig geprüft werden. Vor allem für Mappings von Daten war das ein wirkungsvolles Instrument. Im OSB ist kein Testframework integriert. Es muss auf jeden Fall auf ein zusätzliches Werkzeug zurückgegriffen werden. In der Vergangenheit nutzten wir hierzu immer SoapUI. Allerdings sind das praktisch nur Integrationstests, anstatt Komponententest. Diese Tests lassen sich nicht immer einfach umsetzen, wenn die Eingangs- und Ausgangsnachrichten sehr komplex sind.

Zusätzlich steigerten wir die Qualität durch Nutzung von Werkzeugen zur statischen Codeanalyse. In unserem konkreten Fall nutzten wir PMD und stellten sicher das alle Blocker entfernt wurden. Für den OSB existieren keine Werkzeuge für die Codeanalyse.

Viele Entwicklungsvorgaben im Projekten konnten automatisiert geprüft werden. Ein großer Vorteil war die Möglichkeit die Komponenten einer Schnittstellen unabhängig voneinander testen zu können. Vor allem Mappings änderten sich im Laufe des Projektes häufiger. Durch die automatisierten Tests konnte die Qualität auch bei vielen Änderungen sichergestellt werden.

Für die Entwicklung war der Wechsel enorm lohnenswert. Wie stellte sich das nun für den Betrieb dar?

Es hat sich gelohnt, weil das Betriebsmodell durch Container bestens unterstützt wird.

Die IT hat das Ziel die Verantwortlichkeiten stärker zwischen Plattform und Anwendungen zu trennen. Auf organisatorischer Ebene ist das schon seit geraumer Zeit gelungen. Durch Nutzung einer geeigneten Plattform sollte das auch auf technischer Ebene erreicht werden.

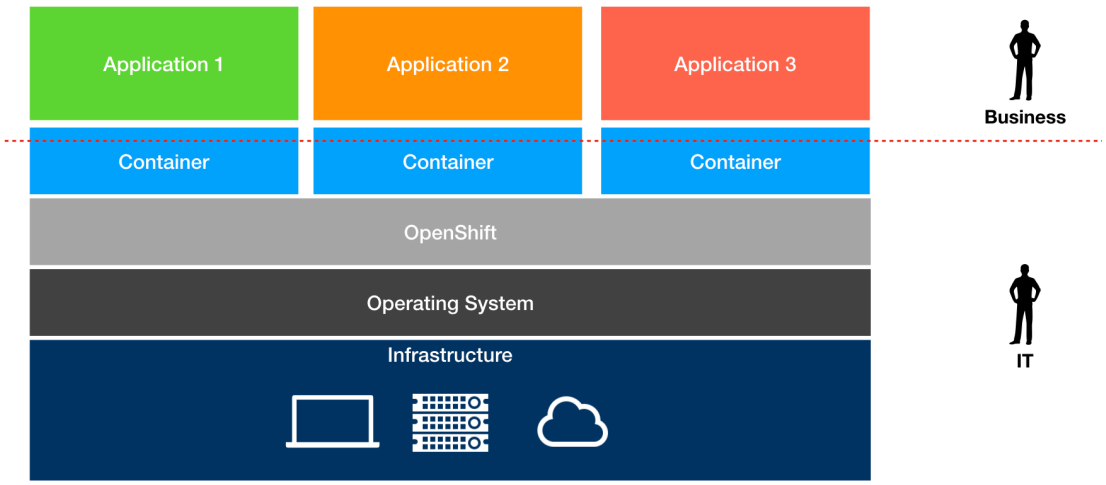

Abbildung 2: Separation of Responsibilities

Mit Containern kann diese Trennung technisch unterstützt werden. Die IT des Kunden verwaltet die Plattform für Container und stellt sicher das diese stabil läuft und bei Bedarf skaliert. Die Verantwortung der Anwendungen in den Containern übernehmen definierte Ansprechpartner aus den Fachabteilungen. Somit ist schon alleine durch Einsatz einer Container-Management-Plattform eine logische und technische Trennung der Verantwortlichkeiten sichergestellt. Insbesondere können durch die Konfiguration von Namespaces (Projekte) die Anwendungen nach fachlichen Gesichtspunkten gruppiert und unterschiedliche Sicherheitsstufen konfiguriert werden. Im OSB ist die Trennung der

Verantwortlichkeiten so nicht darstellbar. Eine logische Gruppierung der OSB-Services nach fachlichen Gesichtspunkten ist ebenso nicht möglich. Obwohl der Enterprise Manager unterschiedliche Rollen kennt, kann jeder Berechtigte im OSB alle Projekte im Dashboard sehen und verwalten. Insbesondere hat jeder Berechtigte Zugriff auf die erstellten Logfiles, was u.U. nicht gewünscht ist. Insbesondere beim Re-Start eines Managed Servers sind immer alle OSB Projekte betroffen. Bei Fuse kann jede einzelne Schnittstelle separat verwaltet werden, ohne andere Schnittstellen zu betreffen.

Fazit

Wir haben erfolgreich die Integrationsstrategie von einem zentralen zu einem de-zentralen Ansatz vollzogen. Die Nutzung von modernen Entwicklungswerkzeugen und -prinzipien hat erheblich zur Akzeptanz beigetragen. Die Qualität konnte enorm durch den Einsatz von Standardverfahren verbessert werden. Ferner werden nur die absolut notwendigen Komponenten für die Lösung eines Problems herangezogen. Insbesondere das Betriebsmodell wird durch die neue Plattform auch von technischer Seite optimal unterstützt. Insgesamt betrachtet gibt es viele Alternativen um das Thema Integration zu lösen. Dabei muss nicht immer zwingend ein ESB oder eine komplexe Integrationsumgebung zum Einsatz kommen.

Kontaktadresse:

Markus Lohn

esentri AG

Pforzheimer Str. 132

D-76275 Ettlingen

Telefon:	+49 (0) 7243 354900
Fax:	+49 (0) 7234 3549099
E-Mail	markus.lohn@esentri.com
Internet:	www.esentri.de