03-2016 | Herbst | www.ijug.eu

Java aktuell

Praxis. Wissen. Networking. Das Magazin für Entwickler

Java aktuell

D: 4,90 EUR A: 5,60 EUR CH: 9,80 CHF Benelux: 5,80 EUR ISSN 2191-6977

Java wächst weiter

Microservices
Schnell und einfach implementieren

Container-Architektur
Verteilte Java-Anwendungen mit Docker

Java-Web-Anwendungen
Fallstricke bei der sicheren Entwicklung

4 191978 304903 03

iJUG Verbund

Neues von den letzten Releases

Sich schnell einen Überblick über bestehende Strukturen und deren Beziehungen verschaffen

Daten in unterschiedlichen Formaten in der Datenbank ablegen

Canary-Releases mit der Very Awesome Microservices Platform

Bernd Zuther, codecentric AG

Immer mehr Unternehmen zerschlagen ihre Software-Systeme in kleine Microservices. Wenn das passiert, entstehen mehrere Deployment-Artefakte, was nicht nur das Deployment des Gesamtsystems komplexer macht. Um diese Komplexität beherrschen zu können und die Auslieferungsmöglichkeiten einer Software zu verbessern, ist der Einsatz von Werkzeugen zur Infrastruktur-Automatisierung unumgänglich.

Nur Unternehmen, die es schaffen, sich schnell genug an die sich ändernden Ansprüche und Anforderungen ihrer Kunden anpassen zu können, werden auf lange oder kurze Sicht überlebensfähig bleiben. Die Absatzmärkte der meisten Firmen erfahren seit einigen Jahren einen drastischen Wandel. Durch Globalisierung, Marktsättigung und das Internet haben sich stark konkurrierende und dynamische Märkte entwickelt, die bedarfsgesteuert sind. Daher sollten sich Unternehmen an die veränderten Bedürfnisse ihrer Kunden schnell anpassen: An dieser Stelle sollte von „kosteneffizienter Skalierung" zu „Reaktionsfähigkeit" gewechselt werden *[1]*.

Canary-Release

Um das Risiko beim Ausrollen einer neuen Software-Version zu vermindern, benutzen viele Unternehmen eine Technik, die sich „Canary-Release" nennt. Dabei werden Änderungen erst einmal nur einem Teil der Benutzer zu Verfügung gestellt und es wird gemessen, wie dieser Teil auf die Änderungen reagiert, bevor man die neue Software auf der kompletten Infrastruktur ausrollt. Durch dieses Vorgehen wird eine Möglichkeit geschaffen, Software schrittweise aus fachlicher Sicht zu verbessern *[2]*.

Abbildung 1 stellt den Aufbau eines Canary-Release dar. Es ähnelt in der Ausführung einem Blue-Green-Deployment *[3]*, das aber das Ziel verfolgt, die Downtime einer Applikation zu minimieren. Bei beiden Verfahren werden zwei Infrastruktur-Stränge benötigt, auf die zwei unterschiedliche Software-Versionen geliefert werden. Dabei handelt es sich jeweils um die neue und die alte Version der Software. Über einen Router werden die Benutzer auf die entsprechende Version umgeleitet.

Bei einem Blue-Green-Deployment macht man einen harten Wechsel auf die neue Version der Software. Beim Canary-Release werden dagegen beispielsweise nur fünf Prozent des Traffics auf die neue Version umgeleitet. Damit soll herausgefunden werden, ob eine neue Funktionalität wirklich eine signifikante Veränderung im Verhalten der Benutzer auslöst. Diese Release-Form kann auch benutzt werden, um A/B-Testing zu implementieren. Das Durchführen von Canary-Releases ist bei einer Microservice-Architektur eine nicht triviale Aufgabe.

Microservice-Architektur

Abbildung 2 zeigt ein Verteilungsdiagramm einer Microservice-Architektur, das die Komplexität dieser Architektur-Form verdeutlichen soll. Hier sind unterschiedlichste Artefakte zu sehen, die entstehen können, wenn

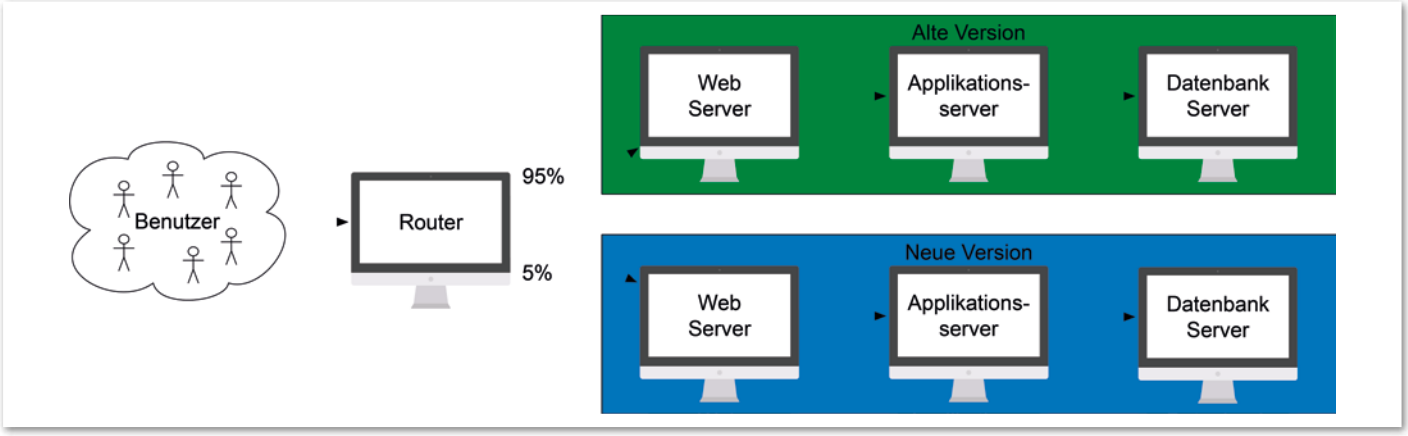

Abbildung 1: Canary-Release

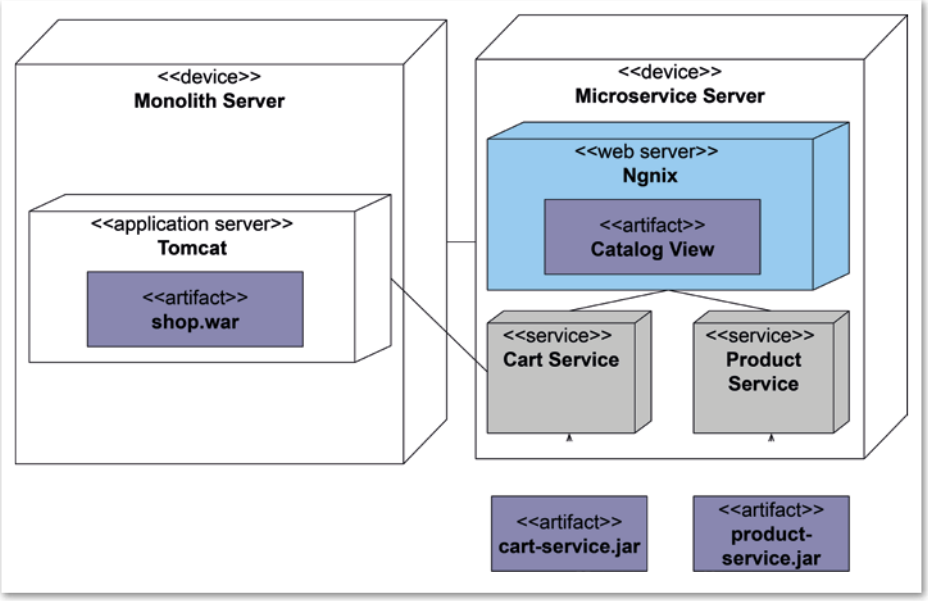

Abbildung 2: Verteilungsdiagramm einer Microservice-Architektur

man eine monolithische Anwendung in eine Microservice-Architektur zerteilt. Auf dem linken Server ist eine Shop-Anwendung zu sehen, die über eine WAR-Datei ausgeliefert wird und über einen Service mit dem rechten Server kommuniziert. Auf diesem Server sind drei verschiedene Anwendungen installiert. Eine Anwendung implementiert eine Katalog-Ansicht, die von einem Nginx ausgeliefert wird. Außerdem sind ein Produkt- und ein Warenkorb-Service dargestellt, die jeweils über eine JAR-Datei gestartet werden. Damit die Anwendung vollständig läuft, sind also vier Artefakt-Arten erforderlich, die unterschiedlich betrieben werden *[4]*.

Docker

Docker eignet sich, um diese unterschiedlichen Artefakt-Arten aus Betriebssicht zu standardisieren. Es bietet Funktionen zum Erstellen und Pflegen von Images, die man über eine sogenannte „Registry" sehr einfach und schnell auf andere Rechner verteilen kann. Betreibt man den Docker-Daemon auf einem Rechner, kann man aus einem solchen Image einen Container erzeugen, der einen laufenden Linux-Prozess beinhaltet. Der Docker-Daemon nutzt zum Ausführen solcher Prozesse spezielle Kernel-Funktionalitäten, um einzelne Prozesse in den Containern voneinander abzugrenzen. Container enthalten kein Betriebssystem, sondern nur die jeweils erforderlichen Linux-Anwendungen und deren benötigte Bibliotheken *[5]*.

Listing 1 verdeutlicht einen Workflow, der mit Docker abgebildet werden kann, um Anwendungen auf beliebigen anderen Rechnern auszuführen. Zuerst wird mithilfe eines Docker-Files und des Build-Kommandos ein Image gebaut. Dann übermittelt man das Image mit dem Push-Kommando in eine private oder öffentliche Registry. Auf dem Zielrechner wird ein Image mit dem Pull-Kommando aus der Registry heruntergeladen und mit dem Run-Kommando gestartet. Danach läuft auf diesem Rechner ein neuer Prozess.

Docker im verteilten System

Um zwei Container miteinander per Netzwerk kommunizieren zu lassen, bietet Docker das Link-Konzept an. Beim Verlinken werden IP und exportierte Ports des Zielcontainers als Umgebungsvariable im nutzenden Container bereitgestellt. Der Container, der den Link erzeugt, kann in der Konfiguration der Anwendung auf diese Umgebungsvariablen zugreifen und muss nicht IP und Port direkt verwenden. *Listing 2* zeigt, wie diese Umgebungsvariablen aussehen.

Die Verlinkung funktioniert allerdings nur, wenn sich die Docker-Container auf dem gleichen Server befinden. Wenn Container über mehrere Server verteilt und miteinander verbunden werden sollen, kann das direkt über IP und Port passieren. Möchte man dies per Hand machen, muss man sich auf den unterschiedlichen Rechnern anmelden, die Container in einer bestimmten Reihenfolge starten und die entsprechenden Umgebungsvariablen setzen. Sollten sich IP oder Port der Zielcontainer ändern, müsste der zugreifende Container angepasst und neu gestartet werden. Dieses Vorgehen würde zu einer statischen Umgebung führen und man könnte nicht einfach neue Rechner hinzufügen oder wegnehmen *[6]*.

Very Awesome Microservices Platform

Die Very Awesome Microservices Platform (VAMP, *[7]*) ist ein Werkzeug, das an dieser Stelle ansetzt. Es verwendet aktuell Mesos und Marathon als Container-Manager. Die Plattform wird von der niederländischen Firma magnetic.io *[8]* entwickelt. Mesos *[9]* sorgt für die Ressourcen-Verwaltung im Cluster und ist verantwortlich für das Starten von Docker-Containern in einem verteilten Rechner-Cluster. Marathon *[10]* ist ein Werkzeug, das auf Mesos aufbaut. Mit Marathon kann das Deployment von Microservice-Anwendungen durchgeführt werden. Marathon stellt einen Scheduler dar, der die Verteilung und Ausführung von Docker-Containern steuert *(siehe Abbildung 3)*.

Vamp ist ein Dienst, der sich oberhalb eines Container-Managers wie Marathon ansiedelt und aus mehreren Komponenten besteht.

```
bz@cc1 $ docker build -t zutherb/product-service .
bz@cc1 $ docker push zutherb/product-service
bz@cc2 $ docker pull zutherb/product-service
bz@cc2 $ docker run zutherb/product-service
bz@cc2 $ docker ps
CONTAINER ID IMAGE                COMMAND
87bb5524067d zutherb/product-service:latest "/product-0.6/bin/
```

Listing 1

```
bz@cc $ docker exec 254e40d85a33 env
MONGODB_PORT_27017_TCP=tcp://172.17.0.2:27017
MONGODB_PORT_27017_TCP_ADDR=172.17.0.2
MONGODB_PORT_27017_TCP_PORT=27017
```

Listing 2

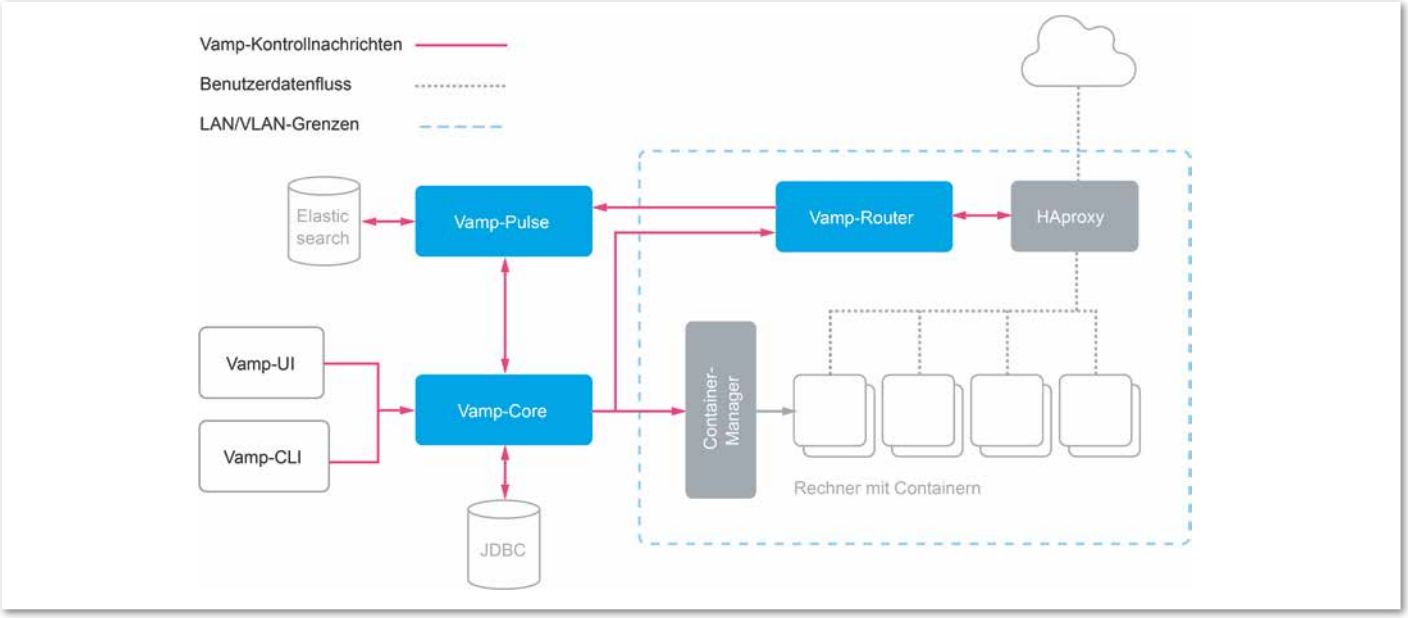

Abbildung 3: Vamp-Architektur

Vamp-Core bietet eine Plattform-agnostische DSL und ein REST-API. Die DSL kann ähnlich wie Giant Swarm *[11]* und Docker-Compose *[12]* benutzt werden, um eine Microservice-Anwendung mit all ihren Abhängigkeiten zu beschreiben und auszuführen.

Listing 3 zeigt einen Ausschnitt aus einer solchen Beschreibung *[13]*. Diese DSL wird von Vamp-Core benutzt, um mit Marathon die Microservices im Mesos-Cluster auszuliefern. Außerdem beinhaltet die DSL Elemente zum Beschreiben von A/B-Tests und Canary-Releases sowie zum Beschreiben von SLAs. Diese SLAs sorgen dafür, dass, wenn bestimmte Antwortzeiten unterschritten werden und noch Ressourcen im Cluster verfügbar sind, automatisch neue Docker-Container im Cluster gestartet werden.

Zum Sammeln der Metriken, die für die SLAs und A/B-Tests benötigt werden, gibt es den Metriken- und Event-Store Vamp-Pulse, in dem Daten von Core und Router in Elasticsearch gespeichert werden. Der Vamp-Router ist eine Komponente, die einen HAProxy steuert, der die Verbindung zwischen den laufenden Docker-Containern und den eigentlichen Benutzern realisiert *[14]*.

Fazit

Um die Reaktionsfähigkeit durch Canary-Releases und A/B-Testing für die Produkt-Entwicklung zu verringern, die Effizienz bei der Auslieferung von Microservice-Architekturen zu verbessern und eine größere Geschwindigkeit in die Auslieferung von Software zu bekommen, ist Vamp eine Plattform, die sich sehr gut für folgende Aufgaben eignet:

- Jeden einzelnen Service in einer Microservice-Architektur erst mal nur im Rahmen eines Canary-Release mit einem kleinen Teil seiner Benutzer testen und feststellen, ob eine Änderung wirklich einen Mehrwert für seine Kunden bringt
- Software schneller und ohne Ausfallzeiten mithilfe eines Blue-Green-Deployment in Produktion bringen
- Testumgebungen einfachen bereitstellen, da auf unterschiedlichen Endpunkten unterschiedliche Anwendungskonfigurationen ausgeliefert werden können
- Bei Lastspitzen sein eigenes Datencenter um Cloud-Infrastrukturen erweitern und automatisch bei bestimmen Antwortzeiten herauf und herunter skalieren
- Daten im Metrik- und Event-System sammeln, um nachhaltige Entscheidungen über Veränderungen treffen zu können. Diese kommen aus dem Live-Betrieb und können vom Anfang bis zum Ende der Wertschöpfungskette vollständig automatisiert betrieben werden.

Quellen

[1] *https://blog.codecentric.de/2015/08/the-need-for-speed-eine-geschichte-ueber-devops-microservices-continuous-delivery-und-cloud-computing/*
[2] *http://martinfowler.com/bliki/CanaryRelease.html*
[3] *http://martinfowler.com/bliki/BlueGreenDeployment.html*
[4] Zuther, Bernd: Microservices und die Jagd nach mehr Konversion., in: Java aktuell (2015), Nr. 2, S. 35 – 40.
[5] *https://public.centerdevice.de/00d60e2d-5490-4df6-afe0-c70824ffb14b*
[6] *https://blog.codecentric.de/2015/08/variation-des-ambassador-pattern-von-coreos/*
[7] *http://vamp.io/*
[8] *http://magnetic.io/*
[9] *http://mesos.apache.org/*
[10] *https://mesosphere.github.io/marathon/*
[11] *https://blog.codecentric.de/2015/05/microservice-deployment-ganz-einfach-mit-giant-swarm/*
[12] *https://blog.codecentric.de/2015/05/microservice-deployment-ganz-einfach-mit-docker-compose/*
[13] *https://github.com/zutherb/AppStash/blob/master/vamp/shop-ab-test.yml*
[14] *https://blog.codecentric.de/2015/10/canary-release-mit-der-awesome-microservices-platform-vamp/*

Hinweis: Das *Listing 3* finden sie online unter: *www.ijug.eu/go/javaaktuell/zuther/listing*

Bernd Zuther
bernd.zuther@codecentric.de

Bernd Zuther ist seit mehreren Jahren als Software-Entwickler tätig. Derzeit ist er bei der codecentric AG als Berater im Bereich Agile Software Factory beschäftigt, wo er vor allem Themen in den Bereichen „Big Data", „Continuous Delivery" und „Agile Software-Architekturen" bearbeitet. Bernd verfügt über langjährige Erfahrung im Bereich E-Commerce und beschäftigt sich seit mehreren Jahren mit der Entwicklung von hochverfügbaren, individuellen Online-Systemen.