Dipl.-Inf. Peter-Christian Quint
(peter-christian.quint@fh-luebeck.de)
ist Wissenschaftlicher Mitarbeiter des CoSA-Kompetenzzentrums der Fachhochschule Lübeck und beschäftig sich im Rahmen des Forschungsprojektes Cloud TRANSIT mit Cloud Computing, Container- und Cluster-Technologien. Der Schwerpunkt liegt im Vermeiden von IaaS-Provider-Abhängigkeiten, insbesondere für KMUs.

Vendor Lock-In im Cloud Computing! Was bringen Container und Container-Cluster?

Der Erfolg von Cloud Computing ist unter anderem der Elastizität und Skalierbarkeit geschuldet. Diese Eigenschaften ermöglichen es, virtuelle Maschinen Last-basiert einem System hinzufügen oder entfernen zu können. Somit können zum einen Lastspitzen abgefangen und zum anderen bei einem kleiner werdenden Bedarf die in Anspruch genommenen Ressourcen (und somit die Kosten) entsprechend verringert werden. Die großen Anbieter bieten diese Services mit eigenen APIs und Diensten an. Daraus resultieren oft Abhängigkeiten, die erst ersichtlich werden, wenn der verwendete Cloud-Dienstleister gewechselt werden soll. In diesem Fall muss eine Anpassung an die äquivalenten Dienste des neuen Providers erfolgen, wenn solche überhaupt angeboten werden. Es stellt sich dann die Frage, ob alle Funktionalitäten überhaupt übernommen werden können und eine Anpassung finanziell und personell möglich ist. Dieser Artikel zeigt, welche Möglichkeiten Container und Container-Cluster darstellen, um eine solche Abhängigkeit zu reduzieren oder sogar zu verhindern.

Infrastructure-as-a-Service

Infrastructure-as-a-Service (IaaS) ist das Fundament von Cloud Computing. Der Name ist sehr treffend gewählt: Der Serviceanbieter stellt die gesamte Infrastruktur. Dazu gehören neben den Servern auch die Netzwerkanbindung, die Stromversorgung, die Kühlung, die Cloud-Software und das nötige Fachpersonal. Dem Kunden wird dies in Form von virtuellen Maschinen (VMs) und Speicherlösungen angeboten. Dabei stehen unterschiedliche Typen zur Auswahl, die sich u. a. in der Anzahl von Rechenkernen, dem Betriebssystem und der Größe des Hauptspeichers und der Festplatten unterscheiden.

Diese Dienste haben für Unternehmen große Vorteile. So ermöglicht IaaS selbst kleinen und mittelständischen Unterneh-

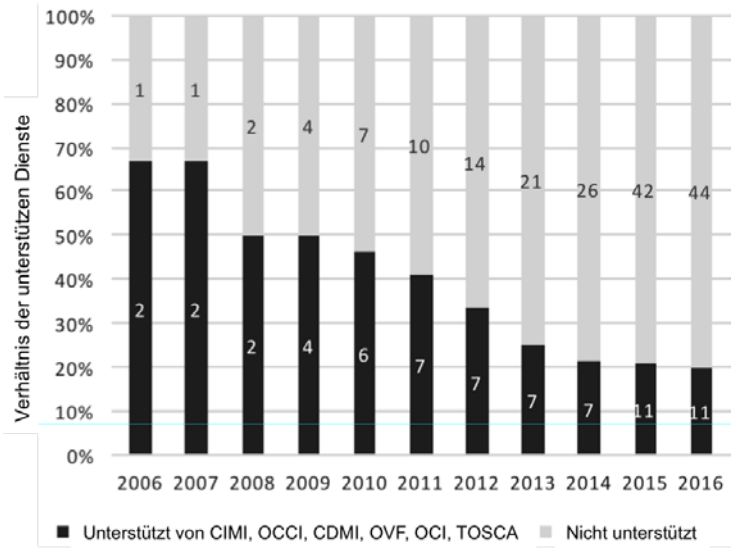

Abb. 1: Unterstützung von standardisierten APIs bei Amazon AWS-Diensten in den letzten zehn Jahren

men Zugriff auf Rechenressourcen, die in großem Umfang bislang nur Großunternehmen vorbehalten waren.

Ein weiterer wichtiger Faktor, der viele Unternehmen zur Nutzung der Cloud animiert, ist die Elastizität, welche es erlaubt, virtuelle Maschinen nach Bedarf in Anspruch zu nehmen. Wenn beispielweise ein Ingenieurbüro eine Simulation mit einem physikalischen Hochleistungsserver in einer Woche (168 Stunden) durchführt, kann in der Cloud mit 168 gleichstarken, virtuellen Maschinen das Ergebnis in einer Stunde vorliegen.

Um die Vorteile der Elastizität auch für permanent laufende Anwendungen wie beispielweise Webdienste zu verwenden, können (automatische) Skalierungsdienste genutzt werden. Diese ermöglichen das last-basierte Allokieren von Ressourcen. So besteht nicht die Gefahr, dass bei einem (plötzlichen) Bedarfsanstieg die Applikation aufgrund von mangelnden Ressourcen ausfällt. Umgekehrt werden nicht benötigte Ressourcen wieder freigegeben, was zu einer Kostenreduzierung führt.

Vendor Lock-In

Man sollte jedoch im Auge behalten, dass beim Verwenden von IaaS-Plattformen Teile der IT an den Provider auslagert werden. Die daraus resultierende Angst vor einer Betreiberabhängigkeit ist bei vielen Unternehmen vorhanden. Wer firmenkritische Bereiche in die Hände von Drittanbietern legt, geht ein gewisses Risiko ein.

Wie Microsoft im Juli 2015 ihre Handysparte, haben in der Vergangenheit sogar große Unternehmen im schlimmsten Fall ganze Produktsparten ersatzlos aufgegeben. Neben diesem Worst-Case-Szenario können selbst eine neue Preisstruktur oder Änderungen an den AGBs für die Kunden negative Folgen haben. Daher wollen viele Unternehmen solche Abhängigkeiten vermeiden.

Im IaaS-Umfeld entstehen diese Abhängigkeiten besonders durch anbieterspezifische und nicht standardisierte APIs für ihre Dienste. Die Grundfunktionen, wie die Bereitstellung von Virtuellen Maschinen, Storage- und Netzwerk-Ressourcen, können oft mit standardisierten APIs genutzt werden. Beispiele für die Standardisierungskonzepte sind das Cloud Infrastructure Management Interface (CIMI), das Open Cloud Computing Interface (OCCI), das Open Virtualization Format (OVF), Cloud Data Management Interface (CDMI) sowie der Open Cloud Initiative (OCI) und OASIS TOSCA.

Dienste, die auf die bereitgestellte Infrastruktur aufbauen, wie beispielsweise Load Balancing und Skalierungsdienste, werden von vielen Providern nicht über diese offenen und standardisierten Schnittstellen zur Verfügung gestellt. In Abbildung 1 ist exemplarisch die Abdeckung von Amazon AWS-Diensten mit solchen Standards zu erkennen. So wurden 2006 etwa zwei Drittel aller AWS-Dienste inhaltlich durch Cloud-Standards abgedeckt. 2016 sind es nur noch etwa 20%. Diese Aussage gilt in ähnlichem Ausmaß auch für alle andere Cloud Service Provider.

Auch wenn bei einem Wechsel äquivalente Dienste vom neuen Service-Provider angeboten werden, müssen wegen des Fehlens von standardisierten APIs entsprechende Anpassungen erfolgen. Besonders bei kleinen und mittelständischen Unternehmen fehlt hierfür oft das entsprechend qualifizierte Personal.

Die Inanspruchnahme externer Firmen, welche die notwendigen Anpassungen vornehmen können, kann unwirtschaftlich sein oder an der Firmenpolitik scheitern. Die Wahrscheinlichkeit, sich von nicht standardisierten (und damit schwer transferierbaren) Diensten abhängig zu machen, steigt von Jahr zu Jahr.

Da oft das Nutzen von Provider-Diensten (beispielweise Load Balancing- und Skalierungs-Dienste) ohne Nutzung der angebotenen proprietären APIs nicht möglich ist, muss zur Vermeidung von Abhängigkeiten ein anderer Weg gewählt werden. Unser Ansatz ist es, die entsprechenden Funktionalitäten und Dienste selbst mit offener und freier Software bereitzustellen. Dies erfordert jedoch entsprechende Architekturvorgaben.

Microservices

Die Microservice-Architektur eignet sich (nicht nur) bei Serveranwendungen, die in der Cloud laufen sollen, durch eine sehr gute Skalierbarkeit einzelner Teile. Die Applikation läuft dabei nicht wie bei der monolithischen Architektur als einziger Block, sondern wird in kleinere Services aufgeteilt, welche jeweils in ihren eigenen Prozessen laufen. Somit sind sie von einander gekapselt und können mit unterschiedlichen Programmiersprachen und Frameworks entwickelt werden.

Auch Hochverfügbarkeit und Skalierung, oft wichtige Eigenschaften von Cloud-Anwendungen, können mit der Microservice-Architektur ermöglicht werden: Im Fall eines Ausfalls von einem Service wird nur dessen Funktionalität beeinträchtig. Im Idealfall hat dies keine Auswirkungen. Wie im Abbildung 2 zu sehen, können Microservices unabhängig voneinander skaliert werden. Im Gegensatz dazu müssen monolithische Applikationen immer als ganzer Block skaliert werden.

Da eine Microservice-Anwendung jedoch aus mehreren Teilen besteht, stellt sich die Frage, wie sie am besten bereitgestellt wird. Während ein Monolith meistens auf einen einzelnen Server installiert werden kann, erfordert jeder einzelne Microservice oft eine eigene, spezifisch konfigurierte Umgebung.

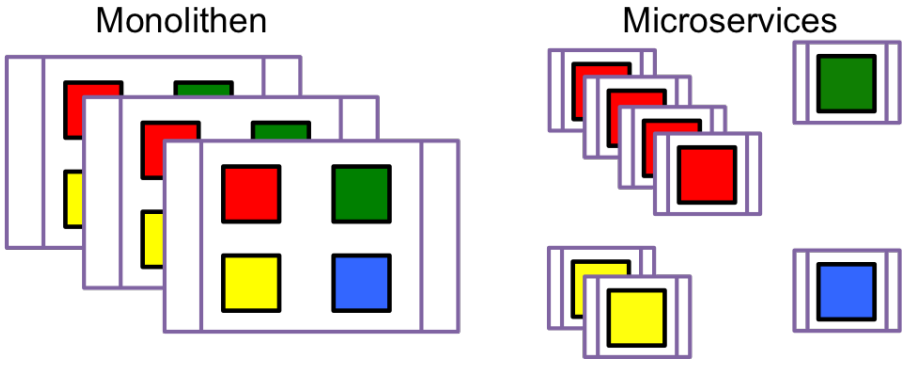

Abb. 2: Unterschiede bei der Skalierung von monolithischen Anwendungen und Microservices

Container für Microservices

Um Microservices nutzen zu können, kann man sie in einer virtuellen Umgebung laufen lassen. Selbst für eine kleine Anwendung werden jedoch viele VMs gebraucht. Durch die Virtualisierung ganzer Maschinen wird jedoch viel Overhead erzeugt. Eine sinnvolle, wenn auch nicht zwingend notwendige, Alternative zu

VMs stellen Docker-Container dar. Durch Nutzung eines gemeinsamen Kernels benötigen sie im Vergleich zu VMs weniger Ressourcen und haben eine sehr kurze Startzeit, die meistens im Sekundenbereich liegt (siehe Abbildung 3).

Applikationen werden in Docker-Images gepackt. Aus diesen können auf unterschiedlichen Maschinen die Container instanziiert werden. Dies kann auf unterschiedlichsten Plattformen erfolgen. So kann das gleiche Image auf einem Entwicklungsnotebook, auf einem Hochleistungsserver oder einer in der Cloud bereitgestellten VM verwendet werden. Anstatt eine Applikation auf einer (virtuellen) Maschine zu installieren, kann dies zudem auch auf einem Container-Cluster erfolgen.

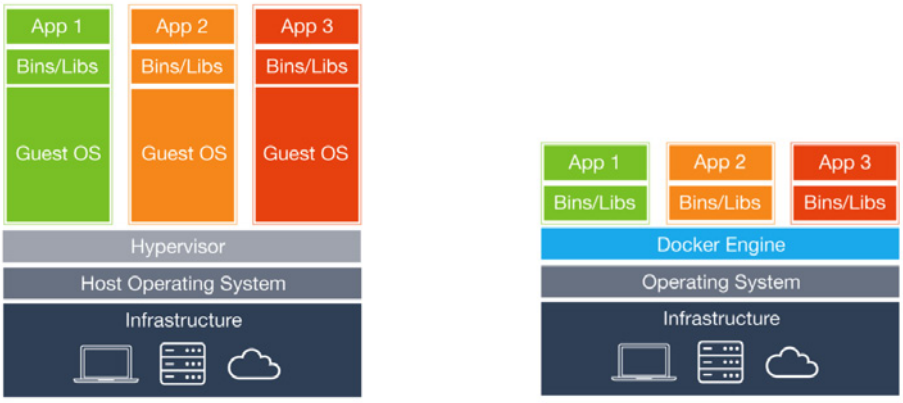

Abb. 3: Virtuelle Maschinen vs. Docker-Container (siehe [Docker])

Container-Cluster

Cluster-Manager bieten Funktionalitäten wie das automatische Skalieren und Erstellen von Containern und anderen Services auf bis zu zehntausenden von (virtuellen) Host-Servern. Zwei wichtige Typvertreter sind Docker Swarm und Kubernetes von Google (weitere interessante Cluster-Manager, auf die hier jedoch nicht weiter eingegangen wird, sind Fleet von CoreOS und Apache Mesos).

Aktuell gibt es mehrere verschiedene Manager mit unterschiedlichen Funktionsschwerpunkten. Diese Cluster-Manager können teilweise auch sinnvoll miteinander kombiniert werden.

Docker Swarm und andere **Cluster Scheduler** fassen mehrere Container-Host-Systeme zu einem logischen Cluster zusammen. Container können dann auf diesem einen logischen Cluster ausgebracht werden. Dabei nutzt Docker Swarm die bekannte Docker API, was die weitere Verwendung von vorhandenen Tools ermöglicht und eine Migration vom Docker Daemon zum Docker Swarm äußerst einfach gestaltet.

In der Version 1.0 wurde Swarm bereits mit tausend Host-Systemen und 50.000 Containern getestet. Ein einfacher Scheduler ist integriert, für komplexere Applikationen können Orchestrierungslösungen wie Kubernetes oder Mesos/Marathon genutzt werden.

Kubernetes dient zum **Orchestrieren** von Docker-Containern. Der Anwender braucht nur noch die Images in sogenannten „Pods“ zu definieren. Ein Pod ist dabei in der Regel eine Sammlung von auf einander aufbauenden Containern, die als kleinste Einheit angesehen werden.

Kubernetes ist nun ohne weiteren Handlungsbedarf seitens des Administrators in der Lage, die Container auf die vorhandenen Ressourcen zu verteilen. Mittels eines „Replication Controllers” kann die Anwendung dabei horizontal skalieren, in dem mehrere Instanzen eines Containers erstellt werden. Im Fehlerfall instanziiert Kubernetes dann fehlende Pods auf noch zur Verfügung stehenden Ressourcen, somit kann ein Totalausfall der Applikation verhindert werden.

Während Container-Cluster häufig aus Gründen der Orchestrierung eingesetzt werden, ermöglichen sie aber auch eine häufig übersehene Reduktion von Betreiberabhängigkeiten (Vendor Lock-In). Beim vorgeschlagenen Einsatz von Container-Clustern wird von den IaaS-Providern nur noch das reine (und gut standardisierte) Bereitstellen der Virtuellen Maschinen genutzt. Häufig nicht durch Standards abgedeckte Funktionen und Dienste wie beispielweise Load Balancing und Applikations-Skalierung werden durch Container-Cluster bereitgestellt und nicht durch den Cloud Service-Provider erbracht.

Ein großer Vorteil, der das Wechseln von Cloud-Anbietern ermöglicht, ist, dass Cluster mehrere Host-Systeme zu einer logischen Einheit verbinden. Dabei müssen die Host-Systeme nicht von einem einzelnen Infrastrukturanbieter bereitgestellt werden. In Abbildung 4 ist exemplarisch illustriert, wie ein Container-Cluster, bestehend aus VMs von unterschiedlichen IaaS-Providern, aussehen könnte. Dies stellt eine gute Basis dar, um zwischen verschiedenen Cloud-Plattformen migrieren zu können.

Multi-Cloud / Migration

Für einen Wechsel oder für die Verwendung von mehr als einer Cloud-Plattform müssen dem Cluster VMs von verschiedenen Anbietern zur Verfügung gestellt werden. Um ein Cluster über mehre Cloud-Plattformen zu installieren, sollten sich die verwendeten VMs unterschiedlicher Provider stark ähneln.

Um stark ähnelnde Paare von Maschinentypen über verschiedene Cloud-Service-Provider identifizieren zu können, kann z. B. EasyCompare verwendet werden. Dies ist ein Benchmarking-Tool, welches im Rahmen des Forschungsprojekts Cloud TRANSIT an der FH Lübeck entwickelt wurde [CoSA1]. Der Cluster ist somit in der Lage, Container auf die Host-

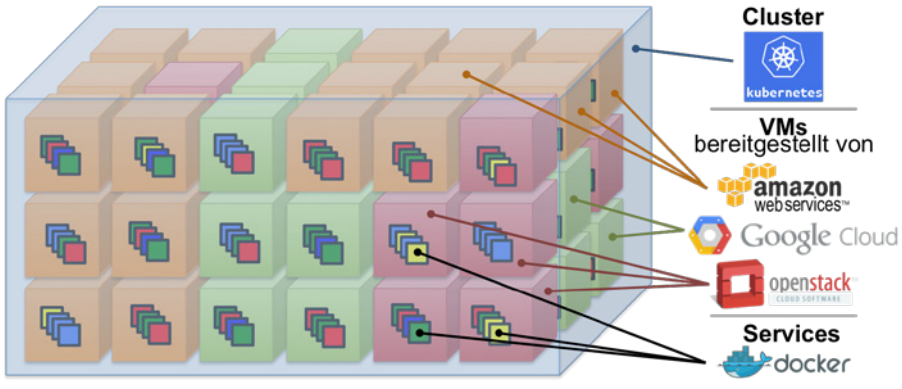

Abb. 4: Container-Cluster aus VMs von verschiedenen Cloud-Plattformen

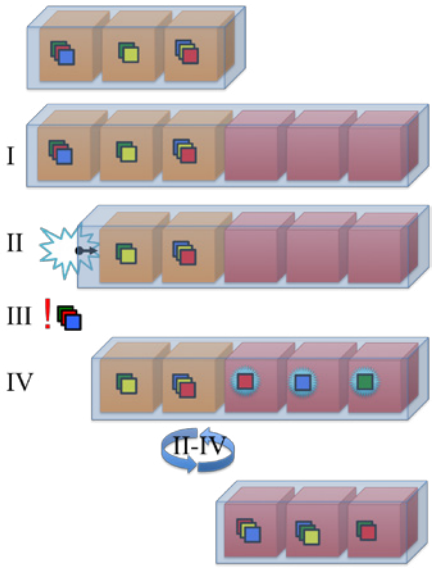

Abb. 5: Migration des IaaS-Providers

Server zu verteilen, unabhängig von dem darunterliegenden IaaS-Provider.

Beim Einsatz eines Container-Clusters kann man einen Wechsel des IaaS-Anbieters recht einfach bewerkstelligen. Wie in **Abbildung 5** illustriert, werden zunächst Maschinen bei einem Zielprovider gestartet und dem Cluster hinzugefügt (Schritt I). Anschließend wird eine VM vom Quell-Provider terminiert (Schritt II). Der Cluster-Manager erkennt das Fehlen von Container (Schritt III) automatisch.

Im Beispiel von Kubernetes beginnt nun der „Replication Controller" die fehlenden Container zu replizieren (Schritt IV). Der Trick ist, dass dies zu den regulären Orchestrierungsaufgaben des Clusters gehört, welche Cluster wie Kubernetes von Haus aus anbieten. Das Terminieren und Replizieren wird so lange sequenziell wiederholt bis alle VMs vom ursprünglichen Provider terminiert sind und die Migration somit abgeschlossen ist.

Container sind idealerweise zustandslos. Wenn diese dennoch einen persistenten Zustand benötigen, müssen die entsprechenden Informationen außerhalb des Containers gespeichert werden, beispielweise in einer Datenbank. Für dieses Szenario muss eine andere Migrationsstrategie angewendet werden.

Eine Möglichkeit ist das Installieren einer neuen Datenbank auf der Zielplattform. Die Container auf der Quellplattform werden nun terminiert. Im nächsten Schritt kann die Datenbank mit den letzten, korrekten Zuständen der Container auf die Zielplattform kopiert werden. Dementsprechend können nun die Container erneut instanziiert werden.

Diese verwenden die neue Datenbank mit den zuvor kopierten Zuständen der terminierten Container. Entsprechend haben die neuen Container den gleichen Status. Im Anschluss können die Datenbank und die Virtuellen Maschinen vom Quell-System entfernt werden, da diese nicht mehr benötigt werden. Moderne NoSQL-Datenbanken (wie CouchDB, Mongo DB, Cassandra, etc.) sind häufig von sich aus bereits auf solche Fehlersituationen (Teilausfälle einzelner Nodes) ausgelegt und erledigen diese Schritte bereits beim sogenannten Sharding.

Fazit und Ausblick

Der hier beschriebene Ansatz stellt eine Möglichkeit dar, zukünftige Applikationen transferierbar betreiben und verteilen zu können. Wenn jedoch bestehende Applikationen auf diese Weise installiert werden sollen, müssen sie (wenn sie als solche nicht vorliegen) in Microservices gegliedert und als Container bereitgestellt werden können. Dieses Vorgehen ist grundsätzlich für alle hochskalierenden Cloud-Applikationen zu empfehlen.

Die Verwendung von quelloffenen Cluster-Managern ermöglicht einen IaaS-Plattform-Wechsel und vermeidet Provider-Abhängigkeiten. Das erfordert jedoch eine Änderung der Betrachtungsweise: Anstatt den Schwerpunkt nur auf die Applikation selbst zu legen sollte diese gleichsam mit einer entsprechende Laufzeitumgebung für die Cloud (dem Container-Cluster) betrachtet werden.

Das Forschungsprojekt Cloud TRANSIT (siehe [CoSA2]) analysiert wie IaaS-Provider-Abhängigkeiten vermieden werden können. Dabei wird zugleich untersucht, wie vorhandene Container- und Cluster-Technologien so gekapselt werden können, dass die damit einhergehende Komplexität auch für Unternehmen mit kleinen IT-Abteilungen (im Extremfall bestehend auch nur aus einer Person) beherrschbar sind.

Interessierte Unternehmen haben die Gelegenheit Ende 2016/Anfang 2017 entsprechende Forschungsprototypen zu testen. ■

Referenz

[CoSA1] https://cosa.fh-luebeck.de/de/cloud/werkzeuge

[CoSA2] https://cosa.fh-luebeck.de/cloud-transit

[Docker] https://www.docker.com/what-docker