Cloud-Architekturen: Klassische Probleme – neu gelöst

Eberhard Wolff, innoQ Deutschland GmbH

Cloud ist mehr als nur ein Deployment-Modell: Es ist die Grundlage moderner Software-Entwicklung. Die effektive Nutzung der Cloud ist der Grund, aus dem Google und Amazon Software so effizient entwickeln können. Ursprung ist ein anderer Architektur-Ansatz – der auch zukunftsweisend für andere IT-Systeme sein kann.

Für den Begriff „Cloud" gibt es viele unterschiedliche Definitionen. Die wesentliche Eigenschaft von Cloud ist Self Service. Ressourcen aus der Cloud kann ein Nutzer einfach buchen – ohne dass dazu irgendjemand manuell eingreifen muss. Konkretes Beispiel: Ein virtueller Server in einer normalen IT-Landschaft ist nur mithilfe eines manuellen Prozesses möglich. Zu den manuellen Schritten kann neben der Einrichtung der Maschine und der Installation der Software auch ein Genehmigungsprozess gehören.

Eine Cloud-Umgebung bietet ein Portal, in das sich der Mitarbeiter einloggen kann. Anschließend startet er dort den Rechner mit einer bestimmten Basis-Installation. Im Hintergrund findet die Abrechnung statt. Statt des Portals kann ein API dem Starten und Stoppen der Maschinen dienen. Dann lässt sich beispielsweise als Reaktion auf eine hohe Last einfach eine neue virtuelle Maschine starten, die dann einen Teil der Last übernimmt.

Welche Ressourcen gebucht werden können, ist ein wesentliches Merkmal zur Unterscheidung der verschiedenen Cloud-Spielarten:

- *Infrastructure as a Service (IaaS)*
  Bietet Infrastruktur wie virtuelle Server oder Storage-Systeme. Beispiele sind Amazon EC2 (Server) oder EBS (Storage) *[1]*. Andere Anbieter wie Digital Ocean *[2]* sind auch in diesem Markt aktiv. Im eigenen Rechenzentrum können solche Lösungen durch Produkte wie OpenStack umgesetzt werden.

- *Platform as a Service (PaaS)*
  Erzeugt eine Infrastruktur, in denen Anwendungen eingerichtet werden können. Der Entwickler muss der Versionskontrolle nur noch Änderungen übergeben. PaaS liest die Änderungen aus und bringt die neue Version automatisch in Produktion. Öffentliche Clouds in diesem Bereich sind Amazon Elastic Beanstalk *[1]* oder Heroku *[3]*; für das eigene Rechenzentrum können OpenShift *[7]* oder Cloud Foundry *[8]* Ähnliches bieten.
- *Software as a Service (SaaS)*
  Bietet Software wie Office Suites oder Customer Relationship Management (CRM) an. Auch Wikis oder Continuous-Integration-Server können in den Bereich „SaaS" fallen. Selbst für Monitoring oder die Auswertung von Log-Dateien gibt es SaaS-Lösungen. Für Software-Entwicklung sind diese Lösungen oft sinnvoll, um eine Infrastruktur kostengünstig und schnell aufzustellen. Die Software-Architektur beeinflusst es aber nicht, sodass dieser Artikel SaaS nicht weiter betrachtet.

Für die Amazon-Cloud-Angebote gibt es ein kostenloses Kontingent für Neukunden. Ein Tutorial führt durch die ersten Schritte *[9]*. Ähnliches gilt für viele andere Clouds wie beispielsweise Heroku *[10]*.

Cloud und Software-Entwicklung

IaaS und PaaS scheinen nur eine Infrastruktur für Systeme anzubieten. Für eine Software-Architektur ist das nicht so wichtig. Schließlich sollte die Plattform, auf der die Software läuft, die Architektur nicht zu stark beeinflussen. Warum sollte sich also die Architektur durch Cloud entscheidend ändern? Die Cloud bietet eine viel flexiblere Infrastruktur. Hardware wird zu Software. Früher bedeutete ein neuer Server, dass ein Admin in einem Server Rack einen neuen Rechner eingebaut hat. In der Cloud ist ein neuer Rechner nur der Aufruf eines API. Dahinter steckt die Idee von „Infrastructure as Code". Auch die Infrastruktur wird also zu Code, der APIs zum Aufbau von Servern aufruft oder Skripte enthält, die Software auf den Servern installieren.

Continuous Delivery

Eine flexible Infrastruktur ermöglicht Optimierungen der Software-Entwicklung. Wenn es so einfach ist, neue Server zu starten und mit Software zu versehen, lassen sich Test-Systeme ohne viel Aufwand aufbauen. Ebenso können Produktionssysteme schnell entstehen. Das kommt dem Continuous-Delivery-Ansatz *[4]* zugute. Grundlage von Continuous Delivery und der Continuous Delivery Pipeline sind die verschiedenen Phasen *(siehe Abbildung 1)*:

- In der Commit-Phase werden die Software kompiliert, die Unit Tests ausgeführt und eine statische Code-Analyse durchgeführt. Diese Phase umfasst also die Schritte, die heutzutage ein Continuous-Integration-Server typischerweise durchführt.
- Die automatisierten Akzeptanztests überprüfen, ob die notwendigen Funktionalitäten korrekt implementiert worden sind.
- Kapazitätstests überprüfen, ob der notwendigen Anzahl Benutzer die erforderliche Performance geboten werden kann.
- Explorative Tests können neue Features oder bekannte Probleme ausleuchten.
- Schließlich wird die Software in Produktion übergeben.

Die Continuous Delivery Pipeline sollte mehrmals pro Tag durchlaufen werden. Dazu sind einige Aufwände notwendig. Die Ansprüche an die Infrastruktur sind sehr hoch. Cloud hilft in verschiedenen Bereichen:

- Für die Test-Phasen kann die Cloud-Infrastruktur ohne große Probleme bereitgestellt werden, was die Durchführung dieser Phasen entscheidend vereinfacht. Die notwendigen Systeme sind auch nur für die Tests notwendig. Das kommt der Cloud entgegen, weil dort nur die tatsächlich verbrauchte Rechenzeit berechnet wird.
- Für die Produktivstellung können verschiedene Ansätze genutzt werden, um das Risiko zu reduzieren. Blue/Green Deployment baut eine komplett neue Produktionsumgebung auf. Erst wenn sich diese Umgebung in Tests bewährt hat, wird auch der Traffic auf die Umgebung umgeleitet. Canary Releasing richtet zunächst Software auf einem Server im Cluster ein. Wenn die Software ihre Funktionstüchtigkeit unter Beweis gestellt hat, kommt sie auch auf die anderen Server. Die dafür notwendigen Infrastrukturen lassen sich mit Cloud-Ansätzen sehr einfach bereitstellen. Sie werden auch nicht besonders lange benötigt. Mit klassischen Ansätzen ist diese Flexibilität kaum zu erreichen. Einfach so eine zweite Produktionsumgebung aufzubauen, ist praktisch unmöglich.

Änderbarkeit

Continuous Delivery nutzt die Flexibilität der Cloud für bessere Deployment-Prozesse. So unterstützen Cloud und Continuous Delivery ein Qualitätsmerkmal einer guten Architektur – nämlich Änderbarkeit. Eine saubere Strukturierung der Software und damit eine gute Software-Architektur sollte die Software ebenfalls leichter änderbar machen.

Continuous Delivery geht Änderbarkeit anders an: Durch die Tests und das geringe Risiko beim Deployment können Änderungen schnell und sicher in Produktion gebracht werden. Dieser Aspekt ist für die Änderbarkeit sehr wichtig. Änderungen an

Abbildung 1: Continuous Delivery Pipeline im Überblick

einem System mit guter Architektur, aber ohne Tests, sind sicher schwieriger als Änderungen an einem schlecht strukturierten System mit vielen Tests. Also bietet Continuous Delivery einen ganz anderen Ansatz für die Änderbarkeit der Software, als die Software-Architektur das tut.

Skalierbarkeit

Ein weiteres wichtiges Qualitätsmerkmal von Software-Architekturen ist die Skalierbarkeit. Typischerweise werden diese Herausforderungen gelöst, indem die Software auf einem oder mehreren Systemen mit der notwendigen Leistung eingerichtet wird. Die Systeme müssen so dimensioniert sein, dass sie auch Lastspitzen handhaben können. Wenn die Vorhersage über die notwendige Leistung falsch ist, ist entweder Geld für Server verschwendet worden oder die Last kann nicht mehr gehandhabt werden – was zu Ausfällen und finanziellen Verlusten führen kann. Außerdem ist die Skalierung grobgranular: Wenn die Software auf zwei Servern läuft, ist eine Skalierung auf drei Servern machbar – das erhöht die Leistung allerdings gleich um fünfzig Prozent und ist wahrscheinlich zu viel.

In der Cloud kann das Problem gelöst werden, indem die Software abhängig von der Last neue Rechner startet und die Last aufteilt. So nutzt das System nur die Ressourcen, die gerade benötigt werden. Eine Vorhersage ist nicht notwendig. Die Skalierung kann feingranularer sein. Dazu nutzt man mehr, aber weniger leistungsfähige Server – also beispielsweise zehn Server für die normale Last. Dann kann die Leistung in Zehn-Prozent-Schritten erhöht werden.

Besonders gut funktioniert der Ansatz natürlich, wenn es um ein Internet-Angebot geht und die Anzahl der Nutzer nur schwer vorab festgestellt werden kann. Aber auch Anwendungen mit weniger Schwankungen profitieren, da eine Vorhersage über die Last nicht mehr notwendig ist und unvorhergesehene Lastspitzen abgefangen werden können.

Die Architektur kann so Skalierbarkeit anders umsetzen. Allerdings muss die Software auch dementsprechend eine Verteilung auf mehrere Systeme ermöglichen – dabei muss auch eine höhere Leistung erreicht werden. Wenn also das System einen Punkt hat, den jede Anfrage durchlaufen muss, entsteht ein Flaschenhals, der letztendlich die Skalierung gefährdet.

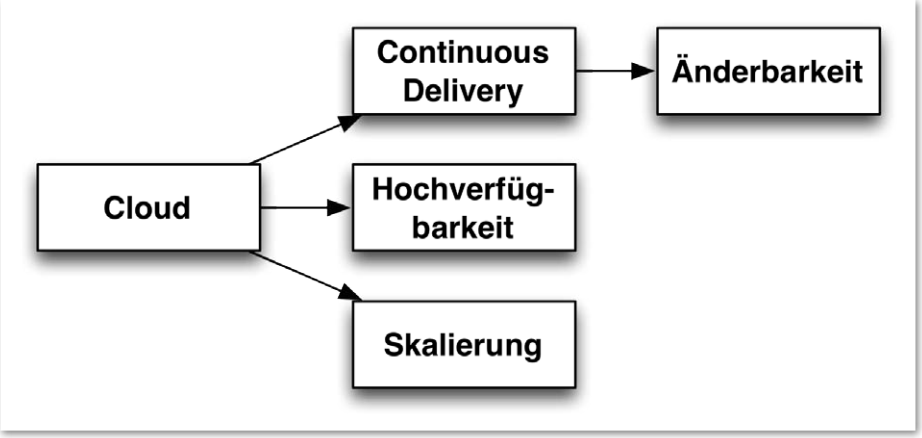

Abbildung 2: Vorteile der Cloud für Software-Architekturen

Hochverfügbarkeit

Klassische Systeme unterstützen Hochverfügbarkeit, indem die Hardware hochverfügbar gemacht wird – beispielsweise durch redundante Netzteile, Netzwerk-Interfaces etc. Zusätzlich wird die Software auf zwei Knoten betrieben, um gegen den Ausfall eines Knotens abgesichert zu sein. Wenn beide Server ausfallen, fällt auch das System aus – daher müssen die Server entsprechend hochverfügbar sein.

In der Cloud lassen sich zur Laufzeit neue Systeme starten. Daher kann der Ansatz zur Hochverfügbarkeit anders aussehen: Die Software startet beim Ausfall eines Knoten einen neuen. Dazu muss die Software mit dem Ausfall eines Knotens umgehen können – also dürfen in dem Knoten keine speziellen Informationen gespeichert sein, die beim Ausfall verloren wären. Dann kann dieser Ansatz sogar eine noch höhere Verfügbarkeit bieten als klassische Architekturen.

In einer klassischen Architektur wird ein großer Teil des Problems von der Software in die Hardware verlagert. Wenn diese ausfällt, funktioniert auch die Software nicht mehr. Dieser Fall tritt vielleicht nicht häufig auf; wenn dies jedoch der Fall ist, sind die Folgen katastrophal. In der Cloud sind Ansätze zum Umgang mit Hardware-Ausfall vorhanden, sodass der Ausfall keine ernsthaften Konsequenzen haben kann.

Die Software so zu strukturieren, dass sie mit dem Ausfall von Hardware umgehen kann, erhöht nicht nur die Verfügbarkeit, sondern senkt auch die Kosten. Schließlich ist keine hochverfügbare Hardware mehr notwendig. Die meisten Cloud-Anbieter haben auch mehrere Rechenzentren, sodass Software sich sogar gegen den Ausfall eines Rechenzentrums oder andere Katastrophen absichern kann, indem dann neue Server in anderen Rechenzentren gestartet werden. Unternehmen wie Google oder Amazon verwenden einen Ansatz für ihre hohe Verfügbarkeit, die sie tagtäglich im Internet unter Beweis stellen.

Warum Cloud?

Viele meinen, dass Cloud vor allem der Reduktion von Kosten dient. Das ist aber nur ein Grund. Vor der Entwicklung der Cloud-Infrastruktur hat Amazon *[5]* beobachtet, dass Entwickler sehr viel Zeit mit Skalierbarkeit verbringen, statt neue Features zu implementieren. Durch die Cloud hat Amazon für sich eine Basis für Skalierbarkeit geschaffen. Die dafür notwendige Software ist ein signifikantes Investment, das durch eine bessere Hardware-Auslastung sicher nicht zu rechtfertigen ist. So bietet die Cloud heute eine andere Basis für Skalierbarkeit und Ausfallsicherheit. Außerdem ist es die Grundlage für moderne Deployment-Verfahren wie Continuous Delivery *(siehe Abbildung 2)*.

Microservices = Architektur für die Cloud?

Um die Vorteile der Cloud auszunutzen, muss allerdings auch die Software passend strukturiert sein. In der letzten Zeit haben sich Microservices *[6]* als neuer Architektur-Ansatz positioniert. Ein Microservices-System ist in einzelne Deployment-Artefakte aufgeteilt, die einzeln und ohne Abstimmung mit anderen Artefakten in Produktion gebracht werden können. In einem E-Commer-

ce-Shop können die Verwaltung der Bestellungen, der Registrierungsprozess und der Katalog jeweils einzelne Microservices sein, die getrennt in Produktion kommen. Eine Änderung am Registrierungsprozess wird also in einem Microservice implementiert, der dann unabhängig von allen anderen Microservices in Produktion gebracht wird.

Jeder Microservice läuft in einer eigenen virtuellen Maschine. Die Kommunikation mit anderen Microservices erfolgt über das Netz mit Protokollen wie REST oder Messaging. So setzen sich Microservices klar von Deployment-Monolithen ab. Dieser Architektur-Stil hat Synergien mit der Cloud:

- Während die Cloud den Aufbau von Infrastrukturen für Continuous Delivery vereinfacht, sind Microservices kleine Deployment-Einheiten. Daher benötigt ein Microservices-System mehr Continuous Delivery Pipelines, die einfacher aufzubauen sind. Ebenso wird der Durchlauf durch die Pipelines schneller, weil für kleinere Deployment-Einheiten auch weniger Tests notwendig sind.
- Die Kommunikation zwischen Microservices erfolgt über das Netz. Jeder Microservice läuft auf einem eigenen Server. Daher ist der Ausfall eines Microservice viel wahrscheinlicher als der Ausfall eines Moduls in einer klassischen Architektur. Microservices müssen also gegen den Ausfall anderer Microservices abgesichert sein. Man spricht von „Resilience“. Beispielsweise können Default-Werte genutzt oder ein vereinfachter Algorithmus verwendet werden, wenn der eigentlich Microservice nicht zur Verfügung steht. Resilience verringert das Risiko des Ausfalls des Gesamtsystems und trägt damit zur Hochverfügbarkeit bei. Außerdem reduziert es das Risiko eines Deployments, da selbst der Ausfall des Microservice nach dem Deployment kaum Konsequenzen hat.
- Die Aufteilung in Microservices kommt der Verfügbarkeit zugute, weil der Ausfall eines Microservice keinen anderen beeinträchtigt – selbst wenn das Betriebssystem oder die ganze virtuelle Maschine in Mitleidenschaft gezogen wird. In einem Deployment-Monolithen kann ein Fehler, der zur Allokation von immer mehr Speicher führt, das gesamte System zum Absturz bringen. Alle Module laufen im gleichen Prozess. Wenn ein Fehler den Prozess zum Absturz bringt, ist das gesamte System ausgefallen. Microservices trennen das System in Module auf, bei denen der Ausfall eines Systems kein anderes System stört. Das ist natürlich auch eine gute zusätzliche Absicherung gegen den Ausfall einzelner Server in der Cloud.
- Microservices können einzeln durch das Starten zusätzlicher virtueller Maschinen skaliert werde. Auch das unterstützt den Skalierungsansatz der Cloud gut.

Fazit

Cloud ist mehr als nur eine Infrastruktur. Ein Architekturziel wie Flexibilität oder Anpassbarkeit kann durch Continuous Delivery und die damit einhergehenden Tests erreicht werden. Diesen Ansatz unterstützt Cloud als Infrastruktur ideal. Verfügbarkeit und Skalierbarkeit lassen sich auch anders umsetzen. Die Aufteilung in Microservices nutzt diese Vorteile ideal aus. So stellt die Cloud die Basis für moderne Architekturen dar.

Weitere Informationen

[1] http://aws.amazon.com/de
[2] https://www.digitalocean.com
[3] https://www.heroku.com
[4] Eberhard Wolff: Continuous Delivery: Der pragmatische Einstieg, dpunkt, 2014, ISBN 978-3864902086
[5] http://jandiandme.blogspot.com/2006/10/jaoo-2006-werner-vogels-cto-amazon.html
[6] Eberhard Wolff: Continuous Delivery: Grundlagen flexibler Software Architekturen, dpunkt, erscheint 2015
[7] https://www.cloudfoundry.org
[8] https://www.openshift.com
[9] http://aws.amazon.com/de/getting-started
[10] https://signup.heroku.com/www-home-top

Eberhard Wolff
eberhard.wolff@innoq.com