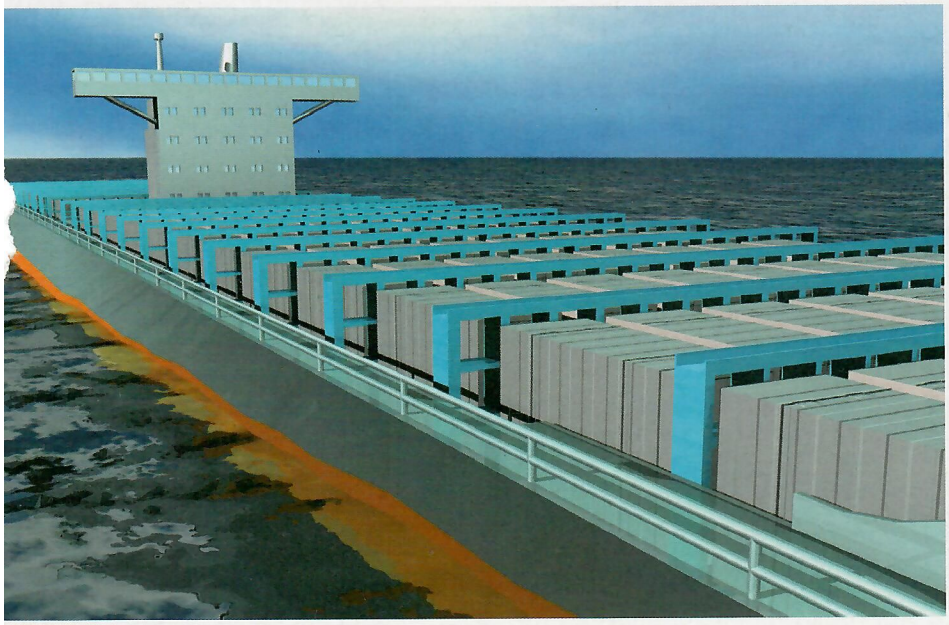


Virtualisierung via CaaS und PaaS

Richtig gestapelt

Christoph Huber, Daniel Takai

Container as a Service und Platform as a Service versprechen in Sachen Virtualisierung von Webanwendungen große Vorteile gegenüber konventionellen Architekturen. Aber was verbirgt sich dahinter, und wann nimmt man was?

Traditionell wird Software per Paketmanager auf dem Betriebssystem installiert, auf dem sich dann die verschiedenen Bibliotheken und Softwarepakete vermischen. Schon lange sind die daraus resultierenden Probleme bekannt. Exotische Fehlerbilder ziehen langwierige Analysen nach sich, die viele Stunden später vielleicht eine minimale Differenz in der Version einer verwendeten Bibliothek oder einen zusätzlichen Slash in einem Konfigurationseintrag aufspüren.

Solche Effekte treten besonders häufig auf, wenn das Betriebssystem und ein Administrator die Bibliotheken auf einem sogenannten Snowflake Server manuell pflegen (siehe Kasten „Müll besser vermeiden"). Ganz schnell wird die Sache bei steigender Anzahl der beteiligten Personen immer chaotischer. Konfigurations- und Automatisierungssoftware wie Ansible oder Puppet versprechen zwar Abhilfe, da sie es ermöglichen, einheitliche Konfigurationen zu verwalten und auszuspielen [1]. Allerdings verlangen solche Werkzeuge fundiertes Wissen und Erfahrung. Sie einzusetzen ist aufwendig und das Entwicklungsteam hat dafür selten Zeit.

Dies endet oft in einer unbefriedigenden Situation: Die Kosten für das Konfigurationsmanagement bleiben trotz guter Werkzeuge hoch, und es entstehen wieder inkonsistente Systemzustände, da nur ein Teil der Konfigurationen aktiv gewartet wird. Das frische Aufsetzen einer Umgebung ist hier gar nicht oder nur mit großem Aufwand möglich. Wer nicht in diese Falle tappen will, sollte um Snowflake Server also einen Bogen machen.

CaaS- und PaaS-Angebote (Container as a Service, Platform as a Service) erfüllen das Versprechen der sogenannten Immutable Infrastructure und verhindern Snowflake Server effektiv (siehe Kasten „In Stein gemeißelt"). Aber was macht diese Virtualisierungen aus, und wie unterscheiden sie sich von anderen? Abbildung 1 zeigt die verschiedenen Schichten, die nun genauer unter die Lupe genommen werden sollen.

Prozesse und Daten sauber trennen

Bei Infrastructure as a Service (IaaS) stellt der Anbieter eine Laufzeitumgebung für virtuelle Maschinen bereit. Der Anwender muss sich also nicht mehr um Hardware und Netzwerk kümmern, aber immer noch um das Betriebssystem und alle darauf aufbauenden Schichten. Auf die Maschinen greift er über einen Hypervisor zu. Der isoliert vor allem die virtuellen Maschinen, damit sich deren Prozesse und Daten nicht in die Quere kommen.

Für die Entwicklung ist eine IaaS-Umgebung genauso wenig hilfreich wie das eigene Blech im Keller: Der Aufwand für das Konfigurieren des Betriebssystems und aller darüberliegenden Ebenen bleibt gleich. Es entfallen lediglich die Kosten für Hardware, Mieten, Strom und Ersatzteillager. Vergleicht man die Preise der IaaS-Anbieter, so stellt man jedoch bisweilen fest, dass man mehr Geld investieren muss als beim Eigenbetrieb. Dazu weiter unten mehr.

CaaS stellt auf Basis eines Image und seiner Konfiguration automatisch Container bereit, verteilt, verwaltet und überwacht sie. Prominentester Vertreter dieser Technik ist Kubernetes [2]. Die Container werden anders als bei IaaS auf einem Betriebssystem virtualisiert. In Abbildung 1 muss man also Betriebssystem und Virtualisierungsschicht vertauschen, um das korrekte CaaS-Schichtenmodell zu erhalten. Wie bei IaaS müssen auch hier die Anwendungen voneinander und vom Betriebssystem getrennt werden. Diese Abgrenzung gilt auch für das Dateisystem,

Müll vermeiden

Ein manuell gepflegter Server sammelt mit der Zeit sogenannten Cruft an. Dieser Müll besteht aus Konfigurationen und Softwarepaketen, die beispielsweise für Bugfixes oder Patches individuell eingespielt wurden. Das erschwert die Administration und benötigt viel Dokumentation. Ein Server, der nicht automatisch per Configuration Service verwaltet wird, heißt Snowflake Server (siehe ix.de/ix1810083).

In Stein gemeißelt

In einer Immutable Infrastructure darf niemand die Konfiguration einer Maschine modifizieren. Sie lässt sich nur ändern, wenn eine neue Maschine hinzukommt. Im laufenden Betrieb dürfen Administratoren keine lokalen Konfigurationen vornehmen. Das bedingt unter anderem auch, dass der Service selbst seinen Zustand externalisieren muss, damit Neuprovisionierungen gelingen können. Beispielsweise sollte er seine Konfiguration in eine Datenbank schreiben und nicht in das lokale Filesystem. Erst danach lassen sich Dienste vollständig automatisch wiederherstellen. Da Webanwendungen häufig neu justiert werden, benötigen sie oft auch neue Maschinen. Somit erreicht die Methode eine hohe funktionale Qualität der Betriebsabläufe. Sie sorgt zudem für Integrität in der Konfiguration und verhindert das Entstehen von Snowflake Servern.

sodass die Container zwischen verschiedenen Systemen austauschbar sind.

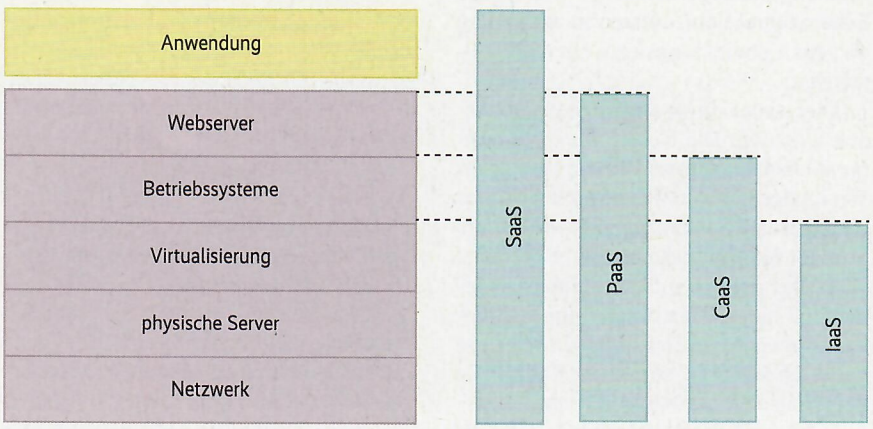
Der größte Unterschied zu einem traditionellen Deployment ist, dass die Laufzeitumgebung als fertiges Image angelegt wird und nicht erst beim Deployment entsteht. Die Instanziierung des Containers entkoppelt die Anwendung von der darunterliegenden Infrastruktur und macht sie portabel. So lässt sich dasselbe (immutable) Image für Funktionstests, Akzeptanztests und in der Produktion nutzen. Zudem ergibt sich so eine klare Aufgabentrennung: Das Betriebsteam kümmert sich nur noch um die Umgebung und soll und darf nicht mehr in die Anwendung hineinschauen.

Hat ein Image den Funktionstest bestanden, kann ein Continuous-Integration-/-Deployment-Service es nach nochmaliger Prüfung automatisch auf die gewünschte Umgebung schieben. Dieses einfache Vorgehen begünstigt eine rasche Evolution der Software bei minimalem Overhead, denn idealerweise reicht nun ein Commit des Entwicklers zum Aktualisieren der Produktionsumgebung. Möglich ist das zwar auch in einer IaaS-Umgebung, allerdings benötigt man mehr Spezialwerkzeuge in der Produktionskette, verursacht also höhere Kosten. PaaS vereinfacht das Ganze noch weiter, denn hier muss man sich nicht mehr um die Konzeption und Entwicklung der Continuous-Deployment-Umgebung kümmern.

Verfügbare Ressourcen richtig nutzen

CaaS bietet zudem die Möglichkeit der automatischen Skalierung einzelner Container, und zwar sowohl horizontal als auch vertikal. Das kann in Kombination mit der fachlichen Dekomposition der Services eine effiziente und genaue Nutzung der verfügbaren Ressourcen ergeben. Hierzu gehören Dienste für Routing und Load Balancing. Stürzt ein Container ab, stellt ihn die Umgebung automatisch wieder her.

PaaS und CaaS bieten ähnliche Funktionen und sind schwer zu unterscheiden. Tatsächlich funktionieren PaaS-Angebote wie Google App Engine auf einer CaaS-Technik wie Kubernetes. PaaS ist also ein Ansatz, der Quelltext annimmt und daraus eine fertige Laufzeitumgebung baut, installiert und betreibt. Er beinhaltet sämtliche CaaS-Funktionen, erweitert um mindestens das Erstellen der Container auf Basis reinen Quellcodes.



Schichtenmodell der Virtualisierung: In einer PaaS-Umgebung müssen sich die Entwickler anders als bei der CaaS-Variante nicht mehr selbst um die Laufzeitumgebung kümmern (Abb. 1).

Dieses Prinzip homogenisiert die Laufzeitumgebung und erleichtert es, Systembetrieb und Entwicklung klar zu trennen: Ersterer stellt sicher, dass die Plattform und die darauf laufenden Anwendungen funktionieren, der Entwickler muss sich „nur" um den Quellcode kümmern. Will das Betriebsteam beispielsweise eine Komponente oder eine Laufzeitumgebung aktualisieren, erledigt es das global auf der Plattform, das Aktualisieren der Applikationen geschieht automatisch, ohne dass bekannt sein muss, welche Programme wo und wie arbeiten.

Anwendungen, die auf derselben Plattform laufen, lassen sich isolieren, sodass mehrere Entwicklungsteams die Umgebung unabhängig voneinander verwenden können. Programme einer Organisation dürfen nicht auf die einer anderen zugreifen, es sei denn, sie nutzen eine öffentliche Schnittstelle.

Weiterhin kann eine PaaS-Umgebung Datendienste bereitstellen, etwa Datenbank- und Cache-Services. Sie beinhaltet oft eine einfache Konvention, die besagt, wie Applikationen automatisch notwendige Konfigurationen zur Verwendung eines Service erhalten (beispielsweise Verbindungsdaten). Oft lassen sich solche Dienste auf einem Marktplatz mit verschiedenen Attributen bereitstellen. Typische weitere Dienste sind Logging und persistentes Messaging für die asynchrone Kommunikation.

PaaS ist in der Lage, die klassischen Leiden von Webapplikationen gut zu behandeln, wenn sie allein in der Cloud liegen (siehe Kasten „Cloud-native Systeme"). Es ist möglich, sie automatisch mit dem eingehenden Traffic zu skalieren. Datendienste hingegen lassen sich ungleich schwieriger anpassen, je nachdem wie die Anforderungen an die Datenkonsistenz aussehen. Da eine Webanwendung ihren Zustand in einer Datenbank speichern muss, stößt auch hier die Skalierbarkeit an Grenzen. Die Entscheidung für eine Architektur will daher gut überlegt sein, Kapazitätstests für die Dienste sind zu empfehlen.

CaaS und PaaS können mehr

CaaS und PaaS bieten also eine Reihe von Vorteilen:

- Das Trennen von Entwicklung und Systembetrieb schafft klare Verantwortlichkeiten.
- Fehler lassen sich leichter analysieren, weil keine Snowflake Server im Spiel sind.

- Teams arbeiten effizienter, weil Mittelsmänner entfallen.
- Entwickler können die Anwendungen mit Werkzeugen der Hersteller besser handhaben.
- Bei Fehlern ist das Rollback einer Version leichter zu bewerkstelligen.
- CaaS und PaaS sind Türöffner für Testverfahren (Canary Deployments, A/B-Tests, Blue-Green Deployments) [3].
- Das Vorgehen begünstigt Microservice-Architekturen, da das Zusammenstellen von Diensten hier einfacher ist. Denn ohne PaaS ist viel Handarbeit gefordert.
- Dienste lassen sich abhängig von den Anforderungen an die Datenintegrität automatisch skalieren.
- Die Sicherheit erhöht sich, da die Plattform garantiert, dass alle Applikationen die neuesten Sicherheitsupdates verwenden. Bei einer PaaS-Installation geht das sogar bis hin zu den Laufzeitbibliotheken.

Da man sich irgendwann für eine Variante entscheiden muss, stellt sich die Frage nach den wesentlichen Kriterien. Eine Barebone-Lösung auf Basis von IaaS ist heute aufgrund der wirtschaftlichen und technischen Nachteile nicht mehr vertretbar. Wenn keine guten Argumente für den Eigenbetrieb von Hardware vorliegen (Business Continuity, Compliance et cetera), empfiehlt sich der Einsatz einer externen Plattform. Die lässt sich mit einem motivierten und kompetenten Betriebsteam sogar selbst betreiben.

Als Faustregel kann gelten: Komplexe Architekturen mit vielen beteiligten Teams, geografisch verteilten Microservices, speziellen Datendiensten und aufwendigen Konfigurationen sind mit CaaS besser bedient, weil hier die Kontrollmöglichkeiten vielfältiger sind. Und die Teams können ihre Produktionsworkflows an die hauseigene Kultur anpassen. In diesem Fall ist ein dediziertes DevOps-Team für die Konfiguration der CaaS-Plattform zu empfehlen.

Wer nur ein einfaches System mit wenigen Services und einer überschaubaren Anzahl von Teams umsetzen will, für den bietet PaaS die bessere Alternative, weil sich die Entwicklung hier vereinfachen und automatisieren lässt. Zudem existieren vorgefertigte Plattformdienste.

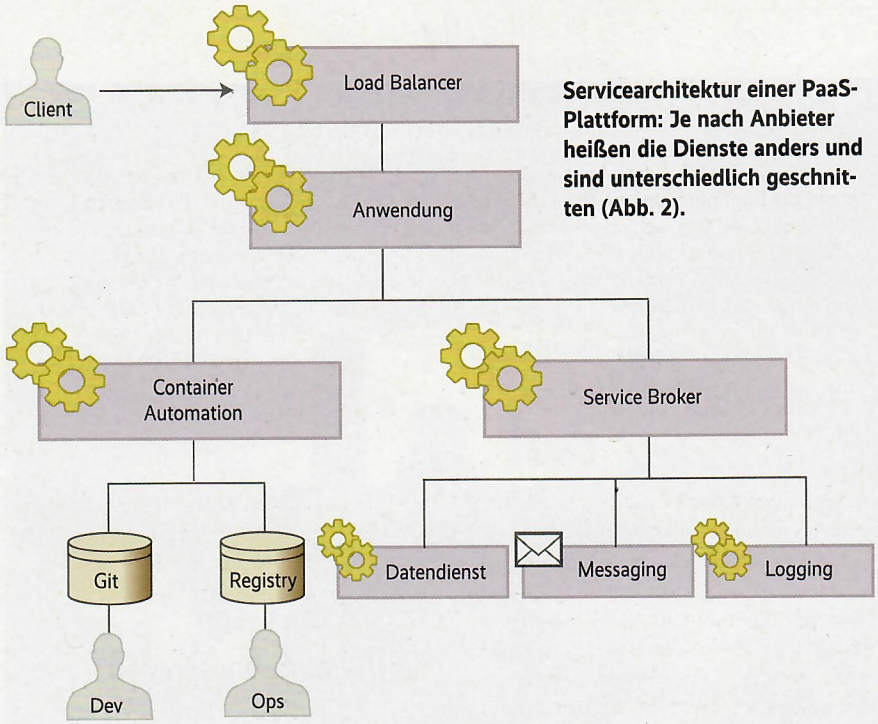
Kosten im Blick behalten

Es ist möglich, Services auf CaaS und PaaS zu mischen. Ein wichtiger Punkt sind die Kosten. Es empfiehlt sich, eine Gesamtkostenbetrachtung anzustellen und nicht nur die reinen Preise der Container pro Stunde mit den Hardwarekosten im eigenen Rechenzentrum zu vergleichen. Der Cloud-Anbieter dürfte in der Regel teurer sein; die Ausgaben für die reine Infrastruktur würden also steigen. Ob das System deswegen insgesamt kostspieliger ist, bleibt zu klären.

Denn der Rechnung sollte man die erhöhte Effizienz im Deployment und das Setup neuer Entwicklungsprojekte zuschlagen. Die Personalkosten für Entwickler bleiben hoch, und schon Einsparungen von wenigen Tagen Arbeitszeit können die CaaS-/PaaS-Rechnung ganz anders aussehen lassen.

Es kommt hinzu, dass beim Einsatz von Microservice-Architekturen, bei denen Anwendungen individuell nach Last skaliert werden können, die Ressourceneffizienz sehr hoch sein kann. Gegebenenfalls sinkt also der Bedarf an Maschinen gegenüber der selbst betriebenen Variante.

Wer heute eine neue Architektur entwirft, kommt um Container nicht herum.



Servicearchitektur einer PaaS-Plattform: Je nach Anbieter heißen die Dienste anders und sind unterschiedlich geschnitten (Abb. 2).

Cloud-native Systeme

Damit ein Service als Cloud-native gilt, muss er einige Qualitätsmerkmale aufweisen:

- Elastizität: Der Service muss horizontal skalierbar sein.
- Herstellbarkeit: Eine Instanz des Service muss sich automatisch herstellen lassen.
- Prüfbarkeit: Es muss möglich sein, Instanzen eines Service leicht zu beobachten.
- Resilienz: Der Service muss Störungen und Ausfälle verbergen und reparieren können und damit hochverfügbar sein.

Ob nun CaaS oder PaaS für das vorliegende Szenario die bessere Wahl darstellt, ergibt sich aus dem Einzelfall. Dabei kann die Entscheidung pro Service und nicht notwendigerweise pro System ausfallen. Dennoch gibt es auch hier einige Indikatoren, die dem Architekten helfen, die richtige Wahl zu treffen. (jd@ix.de)

Christoph Huber

arbeitet bei der Silberrücken AG in Bern als Servicearchitekt. Seine Arbeitsschwerpunkte sind verteilte Systeme, Cloud- und Geschäftsarchitekturen.

Daniel Takai

arbeitet bei demselben Unternehmen als Unternehmensarchitekt, Autor und Facilitator. Seine Schwerpunkte sind sozio-technische Systeme sowie Cloud- und Geschäftsarchitekturen.

Literatur

[1] Victor Volle; Automatisierungs-Tools; Werkzeugkiste; Chef, Puppet und Ansible; Developer-Sonderheft Continuous Integration 2016, S. 70

[2] Erkan Yanar; Clusterverwaltung; Ressourcen fischen; Services mit Kubernetes bereitstellen; *iX* 2/2017, S. 38

[3] Daniel Takai, Architektur für Websysteme; Serviceorientierte Architektur, Microservices, Domänengetriebener Entwurf; Hanser Verlag 2017

Alle Links: ix.de/ix1810083