



Berufsbegleitender Studiengang
Wirtschaftsinformatik, 3. Semester

Hausarbeit
im Rahmen der Lehrveranstaltung
IT-Infrastruktur

über das Thema

**Kubernetes - Eine Revolution bei der Bereitstellung von
Softwarelösungen**

Betreuer : Dr. Arthur Dill

Autor : Carlo Nölle
Matrikelnr. : 492765
Sürther Hauptstraße 47
50999 Köln

Abgabe : 29. Februar 2020

Inhaltsverzeichnis

1	Einleitung	1
1.1	Aktualität und Relevanz	1
1.2	Zielsetzung	1
1.3	Aufbau der Arbeit	1
1.4	Hinweise	1
2	Der Hintergrund	2
3	Aktuelle Möglichkeiten	4
3.1	Bare-metal	4
3.2	Virtualisierung	4
3.3	Kontainerisierung	5
4	Chancen und Risiken	8
4.1	Kubernetes	9
5	Fazit	13

Abbildungsverzeichnis

1	Unterschied der Plattformen	8
2	Kubernetes Logo	9
3	Internetnutzung weltweit	13

1 Einleitung

Diese Arbeit umfasst das Themengebiet rund um die Softwarelösung Kubernetes, welche das Bereitstellen und Pflegen von weiterer Software vereinfacht. Es wird hierbei ein flexibler und skalierbarer Ansatz gewählt der sich von anderen Techniken abhebt.

1.1 Aktualität und Relevanz

Steigende Nachfrage sorgt für immer neue Herausforderungen in der Art und Weise wie eine App (Applikation) oder Website online bleibt. Wer durch Überlastung offline geht verliert potenzielle Kunden.

Kubernetes ist das aktuelle Trendthema für Systemintegranten und Infrastrukturbetreiber und wird von vielen Marktführern eingesetzt. Dadurch und durch die Open Source Lizenz unter dem das Projekt steht ist die Software für viele interessant.

1.2 Zielsetzung

Kubernetes sollte heute an keinem Softwareunternehmen vorbeigehen. Natürlich ist der Aufwand immer dem Nutzen entgegen zu setzen. Mittels dieser Arbeit wird versucht einen Vergleich zu bestehenden Methoden zu ziehen und einen Ausblick auf die Chancen und Risiken des Einsatzes der Software zu geben.

1.3 Aufbau der Arbeit

TODO

1.4 Hinweise

Im weiteren Verlauf dieser Hausarbeit wird aus Gründen der besseren Lesbarkeit das generische Maskulinum verwendet. Weibliche und anderweitige Geschlechteridentitäten werden dabei ausdrücklich mitgemeint, soweit es für die Aussage erforderlich ist.

2 Der Hintergrund

In der Softwareentwicklung gibt es irgendwann den Punkt, bei dem man eine fertige Website oder App programmiert hat. Nun gilt es, dieses Produkt erreichbar zu machen, entweder nur ausgewählten Personen oder gleich der gesamten Öffentlichkeit. Diesen Schritt nennt man Bereitstellung, oder auch zu Englisch Deployment. In dieser Arbeit fokussieren wir uns auf eine Website, die der Öffentlichkeit zugänglich gemacht werden soll.

Diese fertige Website soll über das öffentliche Internet, auch WWW (World-wide-web), von jedem internetfähigen Gerät abrufbar gemacht werden. Dazu benutzt man einen Computer, der die Website bereit hält um sie an Endgeräte auszuliefern. Man nennt solch einen Computer Server, und Endgeräte sind die Clients. Server für Websites nennt man Webserver.

Ein solcher Server hat jedoch harte Grenzen was die Kapazität betrifft. Man spricht von Last auf den Server, welche durch zu viele Clients die gleichzeitig auf der Website sind erzeugt wird. Wann diese Grenzen erreicht sind, hängt davon ab, auf wie viel Hardwareressourcen der Server zurückgreifen kann, also Arbeitsspeicher (RAM) und Prozessor- (CPU) Leistung. Der Server muss die Anfragen verarbeiten können die er erhält, und wenn die Ressourcen schon durch andere Arbeitsschritte ausgelastet sind kann der Server neue Anfragen nicht mehr entgegennehmen.

Eine nicht stark frequentierte Website braucht keinen starken Server, jedoch besteht immer die Möglichkeit, das durch verschiedene Faktoren der sogenannte Traffic, die Menge Daten die über das Netzwerk geschickt werden, rapide ansteigt und somit der Server überlastet wird. Durch nicht beantwortete Anfragen kann bei einem Onlineshop beispielsweise Geld verloren gehen, was natürlich nicht passieren soll.

Ein anderes Problem ist eigentlich ein ganz simples. Wenn die Hardware auf dem der Server läuft einen Defekt hat, ist die Website gar nicht mehr erreichbar. Um dieses Problem zu umgehen dupliziert man einfach den Server auf eine andere Hardwarebasis. Somit hat man zwei Server, die die gleiche Website hosten (bereit halten).

Nur funktioniert das Internet so, dass eine Domain (Url, z.B.: beispiel.de) auf eine Adresse im Internet zeigt. Solch eine Adresse nennt man IP, und sie ist einzigartig im gesamten Netz. Da wir aber jetzt auf eine Adresse zeigen, können somit nicht zwei Server antworten, sondern man landet immer bei dem der die Adresse trägt. Dafür gibt es dann Reverse-Proxy's, oder auch Load-Balancer. Im Grunde machen die nichts anderes, als eine Adresse zu haben, und die ankommende Anfrage an definierte Ziele weiter zu leiten.

Der Name Load-Balancer nimmt den positiven Nebeneffekt schon vorweg, den solch ein Proxy kann eine Anfrage nicht bloß zufällig an eine von n vielen Adressen weiterleiten, sondern auch nach verschiedenen Faktoren agieren. Beispielsweise kann der Load-Balancer sich merken, wie viele Verbindungen gerade auf einem Server laufen, um neue besser an einen Server weiter zu leiten der weniger zu tun hat. Somit sind die zwei Hauptprobleme beseitigt.

Das soeben erklärte Prinzip nennt man Horizontale Skalierung, da ein Server neben den anderen gestellt wird (Horizontal), um somit mehr Last abfangen zu können (Skalierung). Vertikale Skalierung ist sehr ähnlich, man verteilt die Last auf mehrere Server, jedoch nicht auf verschiedene Hardware. Man installiert z.B. zwei Webserver auf einem Computer. Dies kann einen Vorteil bringen was die Lastverträglichkeit angeht, da manchmal eine Software nicht die kompletten Hardwareressourcen alleine benutzen darf, jedoch bringt es keine Ausfallsicherheit.

Durch Horizontale Skalierung erhält man also Ausfallsicherheit, da die Website redundant verfügbar ist, und die Last wird gleichmäßig verteilt auf alle Maschinen, wobei jederzeit weitere eingegliedert werden können. Ausfallsicherheit wird auch Hochverfügbarkeit genannt, da selbst im Falle eines Hardwaredefekts die Website immernoch erreichbar wäre. Wahre Hochverfügbarkeit würde erreicht werden, wenn die Replikate des Servers noch an verschiedenen physikalischen Standorten stehen würden, wodurch man Faktoren höherer Gewalt wie Stromausfällen davon kommt. Infrastruktur, so wird die Hardware/Software Lösung hinter der Website/App genannt, ist ein großes Thema und es gibt viele verschiedene Ansätze die gleichen Probleme zu lösen.

Die Planung, Ausführung sowie Instandhaltung und Pflege von Infrastruktur fällt in den Aufgabenbereich eines Systemintegraten. Zumindest ist dies die klare Trennung zwischen dem Softwareentwickler, also dem Programmierer, und der anderen Hälfte, dem Bereitstellen. Im weiteren Verlauf dieser Hausarbeit werden Lösungswege vorgestellt in denen die Aufgabenverteilung etwas verschmilzt.

Dieses Hintergrundwissen sorgt nun für Verständnis der folgenden Themen.

3 Aktuelle Möglichkeiten

Als Infrastrukturbetreiber, um auch andere Arbeitstitel die heute in der Branche anerkannt sind wie z.B. System Engineer oder Cloud Architect mit einzuschließen, steht man vor verschiedenen Möglichkeiten, wie die Plattform auf der die Software nun bereitgestellt werden soll, betrieben wird. Man möchte Kosten sparen und den Aufwand gering halten. Dazu sind über die Zeit Produkte und Konzepte entstanden die anfallende Arbeit zu erleichtern, zu verbessern. Grundlegend sind die folgenden Konzepte die hauptsächlichlichen Arten der Bereitstellung von Apps wie sie in der Industrie Verwendung finden:

3.1 Bare-metal

Der englische Begriff Bare-metal (zu deutsch: pures Metall) beschreibt die einfache Ausführung einer App oder sämtlicher Software auf einem Server. Alles benötigte wird auf den Server installiert und die App gestartet.

Ein Vorteil hierdurch wäre die Einfachheit. Es gibt keinen Zusatzaufwand oder generelle Strukturelle Änderungen nötig.

Der Nachteil ist jedoch das die installierte App dann den ganzen Server für sich alleine hat. Zwei Optionen tun sich nun auf wenn eine weitere App bereitgestellt werden soll. Entweder einen neuen Server bestellen und in die bestehende Infrastruktur eingliedern, oder die zweite App auch auf den Server mit zu installieren. Ersteres ist mit hohem bürokratischem und zeitlichen Aufwand verbunden, zweiteres könnte in Probleme laufen. Sollte man nämlich mehrere Apps auf einem System laufen lassen, kann es zu Problemen bei gemeinsam genutzten Abhängigkeiten (Softwarepakete) kommen, wenn beispielsweise verschiedene Versionen gebraucht werden.

Dazu kommt der harte Faktor Ressourcen. Das was der Server an Ressourcen besitzt ist geblockt für alles was dort drauf installiert ist, ob das jetzt eine App ist oder mehrere.

3.2 Virtualisierung

Unter Virtualisierung¹ versteht man mittels einer Softwarelösung die Serverhardware die man besitzt effizienter auszunutzen. Sie erlaubt es, auf einem Server mehrere virtuelle Maschinen zu erzeugen, die alle ein eigenes Betriebssystem mit sich bringen.

¹ Vgl. *IT-Agile (2020)*, S., 2020.

Die Controllersoftware kümmert sich hierbei um die Verteilung der zu Verfügung stehenden Ressourcen. Eine virtuelle Maschine wird mit festen Hardwareanforderungen erstellt, und ist dann als eigener Server zu betrachten. Diese virtuellen Maschinen nennt man Gast-Systeme, während der reale Server der Wirt ist. Man kann dadurch viele Probleme umgehen, da man eine Software auf einem eigenen Server installieren kann.

Man reduziert sich auch den großen Aufwand einer Neuanschaffung, denn eine virtuelle Maschine ist für einen Administrator schnell erstellt, während einen neuen Server zu kaufen und in sein Netzwerk einzubauen Wochen dauern kann. Die Effizienz kommt daher das eine Applikation nur soviel Ressourcen verbraucht wie sie muss, da durch die Virtuelle Maschine die App von den tatsächlich verfügbaren Ressourcen nichts mit bekommt und auch nichts blockiert.

Soviel zum Guten, jedoch zieht dies auch Nachteile mit sich. Natürlich ist es ein Schritt vorwärts wenn man es mit dem Einkauf neuer Hardware vergleicht, jedoch ist es immernoch schwergängig seine App ans laufen zu bekommen.

Jede virtuelle Maschine ist wie ein neuer Server. Heißt es muss jedes mal ein neues Betriebssystem installiert und eingerichtet werden. Dazu kommt, das diese Systeme alle auf dem neusten Stand gehalten werden müssen, alleine wegen den Sicherheitsupdates. Um das Neuaufsetzen zu umgehen gibt es zwar mittlerweile weitere Tools, genauso für das Updaten, jedoch bleibt die Schwergängigkeit erhalten.

Sollte der Wirt durch einen Hardwaredefekt ausfallen, sind natürlich auch alle VM's (Virtuelle Maschinen) davon betroffen und fallen im schlimmsten Fall auch aus.

3.3 Kontainerisierung

Die Kontainerisierung ist ein neuerer Ansatz, und baut auf dem Virtualisierungskonzept auf. Der Kernunterschied ist jedoch der, das die verschiedenen Kontexte, bei der Virtualisierung die einzelnen VM's, hier keine komplett eigenen Server darstellen, sondern vielmehr auf dem Betriebssystem des Wirtes aufbauen und darauf eigene Software setzt. Man spricht hier von Containern.

Ein Container beherbergt eine Systembasis und die auszuführende Software, eine App oder Website z.B.. Dabei ist er jedoch wesentlich leichtgewichtiger als eine VM, da der Großteil der benötigten Systempakete vom Wirt kommt. Man erhält sozusagen eine eigene, abgetrennte Maschine mit eigenem Dateisystem, welche teile des Wirtes mitbenutzt.

Dies birgt den großen Vorteil, dass Ressourcen nicht fest an einen Container gebunden werden müssen. Also nicht wie bei VM's, bei welchen man fest blockiert wie viel Ressourcen sie jeweils vom Wirt verbrauchen dürfen. Ein Container startet ohne Angaben von Ressourcen und benutzt einfach flexibel das was er braucht. Dadurch ist die Effizienz noch um einiges höher. Dadurch dass einiges vom Wirt im Container mitbenutzt wird, ist die Startzeit eines Containers um einiges verkürzt. Eine VM muss wie ein richtiger Server hochfahren und alle Teile des Betriebssystems anladen, während ein Container das meiste schon geladen hat bevor er gestartet wird.

Ein weiterer Vorteil durch die Abgabe verschiedener Pakete ist dass der Speicherplatz für diese nicht mehrfach belegt wird. So ist ein Container mit der Linux Distribution (ein Betriebssystem auf Basis des Linux Kernels) Alpine gerade einmal 5 Megabyte groß, während eine VM etwa 130 MB wären.

Wie bei der Virtualisierung werden Container wie VM's von einer Controllersoftware verwaltet. Hierzu gibt es mittlerweile mehrere Alternativen, die größte und bekannteste unter ihnen wäre Docker. Docker bietet seine Software für so gut wie jedes Server Betriebssystem sowie für Desktop Systeme wie MacOS oder Windows an. Ein Container läuft unter jedem Host gleich. Bedeutet wenn ein Entwickler seine App in einen Container gepackt hat, kann er diesen Container genauso auf einem Server sowie einem Kollegen mit einem anderen Betriebssystem zukommen lassen, solange es den Docker Client für das Betriebssystem gibt startet der Container gleich. Dagegen ist eine VM auf einen anderen Wirt zu übertragen in der Realität oft mit höherem Aufwand verbunden, da Netzwerk- und Speicherplatzkonfigurationen z.B. abweichen können.

Während bei einer VM der Server durch die Angabe von Randdaten von der Controllersoftware erzeugt wird, und dann die komplette Einrichtung und Installation von Software erst ansteht, funktioniert das Erstellen eines Containers anders.

Hier vermischt nämlich die Trennung zwischen dem Infrastrukturbereich und dem des Entwicklers. Einen Docker Container erstellt man durch eine einfache Text Datei, in der durch bestimmte Anweisungen Betriebssystem sowie alle nötigen Abhängigkeiten installiert werden, und auch der Startbefehl für die App schon hinterlegt werden. Bei Docker nennt sich diese Konfigurationsdatei Dockerfile. Das Schreiben einer Dockerfile ähnelt dem Schreiben von Code, jedoch stark minimiert auf wesentliche Bestandteile wie: Kopieren von Dateien, Ausführen von Befehlen sowie welchen Ursprung, also Betriebssystem der Container nutzen soll. Nachdem die Angaben gemacht wurden, liest der Docker Client diese Datei aus und erstellt basierend auf ihr ein sogenanntes Docker Image.

Ein Docker Image ist sozusagen der Ordner indem alles für den Container liegt. Wenn man nun einen Container von der App starten möchte, dann gibt man dem Docker Client

den Befehl basierend auf dem Image einen Container zu starten. Dies ist auch der Punkt der es leicht macht Dockercontainer unter verschiedenen Hosts auszutauschen. Dazu ist nämlich nur das Kopieren des Images notwendig, und auf dem neuen Wirt kann dieses einfach gestartet werden.

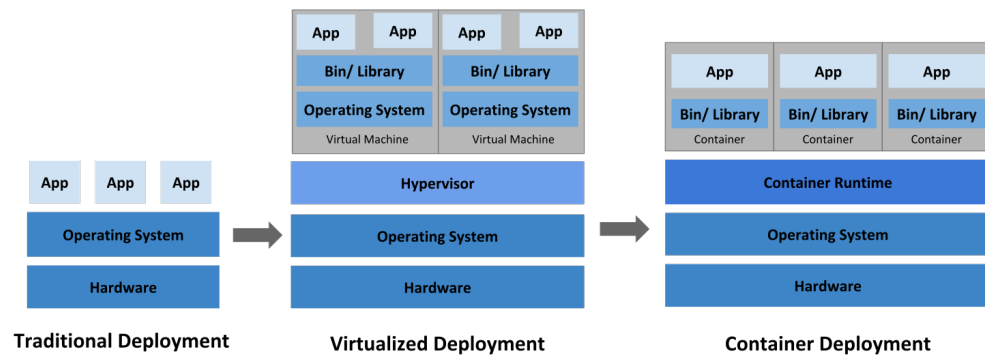
Die Grenzen der Bereiche vermischen nun, da es allgemeine Auffassung, beziehungsweise der angedachte Weg ist das die Erstellung einer Dockerfile in den Aufgabenbereich des Entwicklers, des Programmierers fällt. Denn er weiß genau was für Abhängigkeiten benötigt werden und in was für einen Umfeld (Betriebssystem, auch OS) sein App laufen muss.

Somit fällt das ehemalige VM einrichten und aufsetzen was der Admin machte weg und somit die Bereitstellung in den Bereich des Entwicklers. Man spricht von dem Konzept DevOps, Dev für den Developer (englisch für Entwickler) und Ops für Operations (ein Überbegriff für die Infrastruktur/Admin Seite).

DevOps beschreibt das Zusammenspiel von Entwickler und Admin, und ist kein starres Modell, sondern vielmehr ein Versuch eine Struktur in den Prozess der Bereitstellung zu schaffen. Jedes Unternehmen hat seinen eigenen Ansatz, generell treffen sich die meisten jedoch bei der Ansicht das die Container von den Entwicklern gebaut werden, der Ops diese dann aber irgendwo sicher zum laufen bringen muss. Ob die Ops Seite nur den Server auf dem Docker installiert ist bereitstellen muss und die Anbindung der App an eine Domain wenn es sich um eine Webapp handelt, und der Dev dann seinen Container selber starten und warten muss oder dies anders gelöst ist legt jeder für sich selbst aus.

4 Chancen und Risiken

Abbildung 1: Unterschied der Plattformen



Quelle: <https://bit.ly/32EnzVe>

Mit jedem der genannten Konzepte wird das Ziel verfolgt eine App zu starten sowie diese dann erreichbar zu machen, der Öffentlichkeit oder ausgewählten Personen. Schon genannte Aspekte die zu beachten sind während der Ausfallsicherheit sowie Einfachheit des Arbeitens.

Die erklärten Möglichkeiten der Skalierung während mit allen Konzepten anwendbar, jedoch wird klar dass mittels der Kontainerisierung dies am schnellsten machbar wäre. Während bei Bare-metal oder Virtualisierung ein weiterer Server/VM erstellt und eingerichtet werden muss, muss bei Docker nur ein Befehl ausgeführt werden um innerhalb Sekunden eine zweite Instanz der App hochzufahren. Darüber hinaus ist ein wichtiger Faktor dass bei den Wegen ohne Docker Fehler passieren können. Fehler, die auch vielleicht erst irgendwann während der Benutzung auffallen. Zum Beispiel wenn die falsche Version einer Abhängigkeit installiert wird, oder Dateien an die falsche Stelle kopiert werden.

Diesen Faktor der Fehleranfälligkeit ist bei der Containerisierung komplett irrelevant, denn ein Docker Image ist unveränderbar, und wenn es gestartet wird entstehen immer exakte Kopien.

Zeit ist in jedem Unternehmen eine kostbare Währung. Wie bereits erwähnt zieht man mittels Kontainerisierung in Sekunden eine neue Instanz jedlicher Server hoch. Dies ist mittlerweile ähnlich bei Virtualisierung auch möglich, dort wird mit Kopien von VM's gearbeitet. Bedeutet, dass eine VM mit Betriebssystem präpariert wird, und sobald dann eine weitere Instanz mit dem OS benötigt wird dupliziert man die vorgefertigte Maschine. Dadurch spart man den Schritt der Neuerstellung und Einrichtung, jedoch ist dies in keinsten im Sekundenrahmen, und in den meisten Fällen muss die VM die kopiert werden soll ausgeschaltet sein damit sie kopiert werden kann. Dies würde bei einer Website bedeuten dass

man sie offline nehmen muss, um die Kopierfähigkeit auszunutzen. Bare-Metal hat kaum Möglichkeiten gut ein System zu duplizieren der nicht mit viel händischer Arbeit verbunden wäre.

Soviel dazu, nur muss ja wie schon bereits erwähnt muss um Skalieren zu können ein Proxy vor allen verfügbaren Instanzen stehen, damit eine Adresse auf alle zeigen kann. Dieses Prinzip trifft auf alle Bereitstellungskonzepte gleich zu, jedoch wurde mittels der Software Kubernetes diese Arbeit erheblich erleichtert. Auch die Skalierung wird auf eine neue Ebene gehoben.

Nun sind die hauptsächlichen Chancen und Risiken der verschiedenen Plattformen bekannt, jedoch hat die Containerisierung aufgrund ihrer unveränderlichen Images sowie Geschwindigkeit noch weitere Möglichkeiten die die anderen Plattformen nicht so leicht bieten können.

4.1 Kubernetes

Abbildung 2: Kubernetes Logo



Quelle: <https://bit.ly/3afilMv>

Kubernetes, oder kurz K8s, ist ein von Google gestartetes Open Source Projekt unter der Apache-2.0 Lizenz. Dies bedeutet, dass die Software von jedem verwendet sowie angepasst werden, jedoch nicht unter einer anderen Lizenz weitervertrieben werden darf. Mittlerweile hat der Quellcode, welcher auf der Codeversionierungsplattform Github veröffentlicht ist, über 2400 Mitentwickler. Jeder kann nämlich Anpassungen vornehmen und diese in das Repository, der Speicherort auf Github, einfließen lassen. Dies geschieht jedoch nur, wenn die Anpassung vorher durch verschiedene Tests sowie Abnahmen von anderen Mitarbeitern bestätigt wurde.

K8s baut wie erwähnt auf der Kontainerisierung auf. Im Grunde kümmert es sich um das Verwalten von Containern und übernimmt für uns das Starten und Stoppen. Erstmals ist K8s aber eine Software, die für das Clustering gebaut ist, bedeutet den Verbund von mehreren verschiedenen Servern zu einem. Durch einen solchen Verbund sind nicht nur mehr Ressourcen verfügbar, sondern auch eine gewisse Ausfallsicherheit gegeben. Sollten die einzelnen Knoten, eigene Maschinen im Verbund, jedoch virtualisiert auf einer physikalischen Maschine laufen, kann ein Hardwaredefekt natürlich immernoch alles zum Erliegen bringen. Das Clustering sieht so aus, dass es in einem Verbund eine gewisse Anzahl sogenannter Master-Knoten und eine Anzahl Worker-Knoten gibt. Der Unterschied ist der, dass nur auf den Worker-Knoten die Container, die ausgeführt werden sollen, gestartet werden, während sich die Master-Knoten nur um das Controlling kümmern.

Was hinter diesem Controlling steckt, ist der Kernpunkt von K8s. Ein K8s Cluster kann nämlich erstmal unendlich viele Worker-Knoten, in Zukunft nur noch Nodes genannt, haben. Um diese zu steuern, braucht es mindestens einen Master-Knoten, in Zukunft nur noch Master. Der Master weiß über alle ihm zur Verfügung stehenden Nodes Bescheid, und kann Metriken wie Ressourcenverbrauch abfragen. Die Nodes warten nur auf Befehle vom Master. Für einen hochverfügbaren K8s Cluster benötigt man mehr als einen Master, sowie mindestens zwei Nodes. Nodes nehmen Befehle von allen Masters entgegen, genauso wie alle Master auch im Austausch über ihre Befehle stehen.

Wenn nun eine App, die durch beispielsweise Docker schon containerisiert wurde, gestartet werden soll, gibt man diese Anweisung an einen der Master. Der Master kümmert sich dann darum, dass diese App gestartet wird. Dazu schaut der Master auf welcher seiner Nodes noch genug freie Ressourcen zur Verfügung stehen, und gibt dann den Befehl an die Node weiter, das Docker Image zu starten. Dazu muss das Image auf einen externen Server hochgeladen sein, den jeder der Nodes über das Internet das Image runterladen können. Dazu gibt es sogenannte Docker Registries, in denen man mittels dem Docker Client sehr leicht Images hochladen kann. Damit der Master weiß, wie viele Ressourcen auf der Node frei sein müssen, damit die App überhaupt starten kann, werden Angaben zu Mindestanforderungen sowie auch Höchstanforderungen mitgegeben. Wenn diese nicht mitgegeben werden, startet der Master den Container einfach auf einem zufälligen Node, der noch ein bisschen Platz hat.

Der Vorteil von dieser Weise ist der, dass die Nodes, die dem Cluster angehören, zum einen flexibel hoch- und runterskaliert werden können. Man kann also mit der Anzahl der Apps mitwachsen, und muss nicht von vornherein die ganze Hardware für den K8s Cluster aufwarten. Zweitens können Nodes aber auch an verschiedenen Standorten laufen. Solange die Master mit ihnen über das Internet sprechen können, ist der Standort egal. Heutzu-

tage kann schnell über einen der Cloud-Dienste wie Amazon Web Services oder Google Cloud Platform weitere Hardware auf Minutenbasis gebucht werden. Wenn zum Beispiel ein Onlineshop sich auf einen Ansturm durch eine Verkaufsaktion vorbereiten will, kann er einfach weitere Hardware buchen und seinen K8s Cluster hochskalieren, und nach dem Ansturm wieder herunterfahren.

Die Master übernehmen nicht nur die Verteilung der Docker Container, sie überwachen und messen durchgehend alle möglichen Daten, wodurch weitere Funktionalitäten möglich geworden sind. Beispielsweise das automatische horizontale Skalieren von Docker Containern. Sollte der Master merken das die Last oder der Traffic auf einen Container zu hoch ist, startet K8s für uns einen weiteren Container der gleichen Sorte. Dieser kann wieder auf jeder Node gestartet werden, also auch auf der gleichen wie der erste Container, muss es aber nicht. Wie schon beschrieben sollte nachher aber nur eine Adresse angesprochen werden, um auf einen der Replikat des Docker Containers zu kommen. Dazu hat K8s das eingebaute Feature Services. Ein K8s Service ist ein Verbindungsglied von allen Containern einer Sorte zu einer Adresse. Man kann von einem Loadbalancer sprechen. Dieser Service kann von dem öffentlichen Netz angesprochen werden. Ein Service ist über alle Nodes gespannt, sodass die Replikat des Containers auf jeder Node laufen können, und der Service dennoch alle von ihnen erreicht.

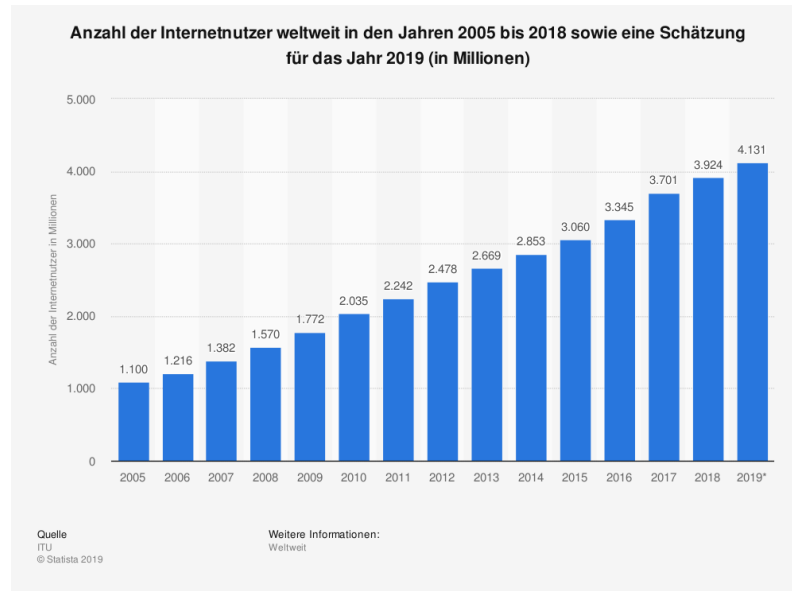
Zusammengefasst kann durch einen K8s Cluster zum einen die wirkliche Hardware flexibel hoch und runter skaliert werden, und das auch regionenübergreifend, und zum anderen können und die K8s Automatismen die Skalierungsarbeit unserer App abnehmen. Wenn man das DevOps Prinzip hier anbringt, dann ist der wirkliche K8s Cluster, also das Aufsetzen und Warten dieses, der Aufgabenbereich der Ops, und das Benutzen dessen die Aufgabe der Devs. Ein Ops hat somit nicht mehr eine wachsende Anzahl einzelner Server die einzeln gepflegt werden müssen, sondern nur noch eine einzige Plattform für alle auszuführenden Apps.

Kubernetes bietet also erhebliche Vorteile gegenüber anderen Plattformen, jedoch ist es auch eine neue Stufe der Komplexität. Ein Ops muss sich in die neue Technologie erst einarbeiten. Dazu zählt erst einmal die Container Engine, zu meist Docker, zu verstehen. Jedoch ist dies auch die kleinste Hürde, den auch wenn das Prinzip von K8s zwar leicht verständlich ist, sieht die Umsetzung etwas komplizierter aus. Dazu kommt auch, das sich die Software ständig weiterentwickelt, und man immer dazulernen muss. Im Jahr kommen mehrere neue Releases raus, wodurch auch der Updateaufwand des Clusters nicht unerheblich ist. Generell kommt zum Überwachen und Warten noch einiges an Software hinzu die ein Ops beherrschen muss um den Cluster ordentlich zu Überwachen. Dies ist jedoch wichtig, denn wenn die Master beispielsweise ausfallen, sind die meisten Dienste die im

Cluster laufen nicht mehr erreichbar, was einem Hardwaredefekt bei einer einzelnen Maschine gleichkommt. Jedoch sei auch gesagt, dass das Clustering einen gewaltigen Vorteil bringt. Sollte eine Node ausfallen, fallen auch alle darauf laufenden Container aus. Jedoch werden die Master dies bemerken, und versuchen auf den anderen verfügbaren Nodes alle Container wieder neu zu starten. Dadurch kommt es maximal zu einem kurzen Ausfall eines Dienstes, und sollte dieser mehrere Replikate auf verschiedenen Nodes haben, kommt es sogar zu gar keinem Ausfall. Ein Ops würde in so einem Fall die kaputte Node entweder reparieren und wieder in den Cluster eingliedern, oder einfach ausschalten und eine komplett neue Node erstellen und dem Cluster anhängen.

5 Fazit

Abbildung 3: Internetnutzung weltweit



Quelle: <https://bit.ly/2TogWCc>

Die IT Branche entwickelt sich stetig weiter. Das muss sie auch um mit dem steigenden Wachstum mithalten zu können.

Während einige Trends schnell in Vergessenheit geraten, etablieren sich wiederum andere als solide Optionen in der Industrie. Grundsätzlich muss jedes Unternehmen abwägen ob es sich lohnt eine neue Software im Betrieb zu etablieren. Da dies immer mit Kosten verbunden ist, alleine als Zeitfaktor, will dies eine überlegte Investition sein.

Dazu sucht man sich Indikatoren an denen man messen kann ob eine Software nur ein Trend ist oder bestand hat. Solche Indikatoren wären zum Beispiel die Zukunftsaussicht der Lösung. Wenn eine Software von einem anerkannten bekannten Unternehmen der Branche stamm, ist das Vertrauen natürlich ein anderes als wenn ein Startup oder eine Privatperson dahinter steht. Auch das Alter der Software und Ehrfahrungsberichte von Unabhängigen haben ein großes Gewicht im Vergleich.

Kubernetes kann einige IT Prozesse im Unternehmen vereinfachen und automatisieren, was Kosten spart. Dennoch muss das Wissen erst aufgebaut werden, somit ist dies eine Rechnung die jedes Unternehmen für sich machen muss. Die Frage ist nach dem ROI, dem Return-of-investment.

Was die Indikatoren angeht, kann man sagen das K8s dort sehr gut abschneidet. Wie erwähnt ist das Projekt von Google gestartet und wird seit je her auch von ihnen weiterentwickelt. Man weiß das Google Kubernetes für ihre eigenen Services einsetzt, und auch K8s als Managed Service, als Software as a Service (SaaS) Lösung in ihrer Cloud Platform anbietet. Darüber hinaus setzen sehr viele der großen Unternehmen auf die K8s Plattform, wie z.B. AirBnB und Reddit. Man kann also davon ausgehen, das die meisten dieser Unternehmen an der Weiterentwicklung von K8s interessiert sind. Somit ist der Fortbestand kein Problempunkt.

Auch auf der Seite der Erfahrungsberichte kann Kubernetes punkten. Kubernetes würde nicht so viel Verwendung finden würden die Unternehmen darin nicht einen Vorteil sehen. Natürlich macht K8s nicht in jeder Umgebung Sinn. Jeder muss abwägen wie er seine Infrastruktur aufbaut. Wenn man weiß das sie aber auf dauer stetig wachsen wird, und man auf Punkte wie Hochverfügbarkeit setzen möchte, sollte Kubernetes auf jeden Fall eine Option sein die angeschaut werden sollte.

Internetquellen

IT-Agile (2020), Softwareunternehmen (2020): Virtualisierung und Containerisierung, URL:
<<https://www.it-agile.de/wissen/agiles-engineering/virtualisierung-und-containerisierung/>> (2020-03-23) [Zugriff:
2020-03-23 20:30 Uhr]

Ehrenwörtliche Erklärung

Hiermit versichere ich, dass die vorliegende Arbeit von mir selbstständig und ohne unerlaubte Hilfe angefertigt worden ist, insbesondere dass ich alle Stellen, die wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen sind, durch Zitate als solche gekennzeichnet habe. Weiterhin erkläre ich, dass die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde/Prüfungsstelle vorgelegen hat. Ich erkläre mich damit einverstanden, dass die Arbeit der Öffentlichkeit zugänglich gemacht wird. Ich erkläre mich damit einverstanden, dass die Digitalversion dieser Arbeit zwecks Plagiatsprüfung auf die Server externer Anbieter hochgeladen werden darf. Die Plagiatsprüfung stellt keine Zurverfügungstellung für die Öffentlichkeit dar.

Köln, 29.2.2020

(Ort, Datum)



(Eigenhändige Unterschrift)