FERDINAND PORSCHE
FERN FH

Kubernetes für Einsteiger

Ein kompakter Guide

Autor: **Bernd Klaus, BA**

***Auszug** aus der Bachelorarbeit:*
„WordPress mittels Kubernetes in der Cloud Eine wirtschaftliche Betrachtung“ ©

Inhaltsverzeichnis

1.1 Kubernetes 4

1.1.1 Einführung 4

1.1.1.1 Geschichte 4

1.1.2 Basis 4

1.1.2.1 Docker 4

1.1.2.2 Aufgaben 6

1.1.2.3 Architektur 6

1.1.2.4 Plug-ins 8

1.1.3 Einsatz 9

1.1.3.1 API 9

1.1.3.2 kubectl 9

1.1.3.3 YAML 9

1.1.3.4 Objekte 10

1.2 Cloud-Provider 17

1.2.1 Public Cloud 17

1.2.2 Provider-Übersicht 18

1.2.2.1 Amazon Web Services 18

1.2.2.2 Microsoft Azure 19

1.2.2.3 Google Cloud 20

1.2.3 Preisgestaltung 21

Literaturverzeichnis 22

Anhang 24

Abbildungsverzeichnis

Abbildung 1: CMS Vergleich 9
Abbildung 2: WordPress Komponenten 11
Abbildung 3: Kubernetes Logo 13
Abbildung 4: Kubernetes Architektur 15
Abbildung 5: Kubernetes Objekte 20
Abbildung 6: Public-Cloud Anteile 26
Abbildung 7: WordPress Standard-Bereitstellung 31
Abbildung 8: Kubernetes Service Erstellung 33
Abbildung 9: Kubernetes Kosten monatlich pro Service 41
Abbildung 10: Fix-Kosten 47
Abbildung 11: Fehlkosten Preisentwicklung 48

Tabellenverzeichnis

Tabelle 1: AWS Preise 28
Tabelle 2: Azure Preise 29
Tabelle 3: Google Preise 30
Tabelle 4: Tatsächliche Kosten 42
Tabelle 5: Kosten je Einheit 44
Tabelle 6: Berechnung - 2 Seiten 45
Tabelle 7: Berechnung - 3 Seiten 45
Tabelle 8: Berechnung - 4 Seiten 46
Tabelle 9: Fixkosten je Anzahl 46
Tabelle 10: Fehlkosten je Anzahl 47

Codeverzeichnis

Code 1: Dockerfile Beispiel 14

Code 2: Deployment 19

Code 3: Passwort-Erzeugung 24

Code 4: Secret 24

1.1 Kubernetes

1.1.1 Einführung

Eingangs möchte ich mit einem kurzen Zitat beginnen, welches die Stärken von Kubernetes beschreibt. Anschließend wird in den weiteren Kapiteln beschrieben, wie Kubernetes entstanden ist, wie die Eigenschaften des Zitats zustande kommen und wie man diese sinnvoll nutzen kann, um seine WordPress-Installation zu hosten.

„Kubernetes möchte jedem Sysadmin danken, der um 3 Uhr in der Früh geweckt wurde, um einen Prozess neu zu starten. Jedem Entwickler, der Code in die Produktivumgebung geschoben hat, um dann festzustellen, dass er dort nicht wie auf dem eigenen Laptop lief. Jedem Systemarchitekten, der unabsichtlich einen Last-test gegen den Produktivservice laufen ließ, weil irgendein Hostname nicht angepasst wurde. Dieser Schmerz, diese unfreundlichen Arbeitszeiten und diese verrückten Fehler haben die Entwicklung von Kubernetes inspiriert.“[1]

1.1.1.1 Geschichte

Bereits 2013 wurde Google-intern das Projekt Borg gegründet, welches das Ziel verfolgte, einen Orchestrierungsdienst für die damals neuartigen Docker Container bereitzustellen. Am 21. Juli 2015 entschloss sich Google, das Projekt der Open-Source-Community unter dem Namen Kubernetes zur Verfügung zu stellen.

Der Name stammt aus dem Griechischen und bedeutet frei übersetzt „Steuermann“. Dementsprechend ist das Logo entstanden, welches das Ruder eines Schiffs symbolisiert.

Abbildung 3: Kubernetes-Logo
Quelle: https://github.com/kubernetes/kubernetes

1.1.2 Basis

1.1.2.1 Docker

Der Kern von Kubernetes ist Docker, dessen Funktionalitäten am besten mit jenen einer virtuellen Maschine vergleichbar sind. Dabei stellt ein Docker Container eine einzelne ausführbare Instanz dar. Im Unterschied zu einer herkömmlichen virtuellen Maschine sollte pro Container immer nur ein einziger Prozess ausgeführt werden. Entsprechend Abbildung 2 sind demnach drei eigenständige Docker-Container für den Betrieb einer WordPress-

[1] (Kelsey Hightower, 2018)

Website notwendig[2]. Diese Methodik folgt einem Microservice-orientierten Ansatz, welcher auch in zahlreichen Best Practices rund um Kubernetes festgehalten ist. Im Unterschied zu einer virtuellen Maschine könnten alle diese Teile (*Abbildung 2*) simultan betrieben werden. Der Vorteil, welchen Docker an dieser Stelle bietet, ist, dass pro Host mehrere idente Container parallel laufen können. Dabei beeinflussen sich die Systeme selbst nicht, da die Container jeweils in ihrer eigenen Sandbox verankert sind und somit vollständig voneinander isoliert ausgeführt werden. So können auf ein und demselben Host unterschiedliche WordPress-Versionen, welche im Weiteren auf unterschiedliche PHP-Versionen angewiesen sind, konfliktlos nebeneinander betrieben werden. In einer Ausprägung ohne Docker stellt die Ausführung von unterschiedlichen PHP-Versionen auf demselben Host einen immensen Aufwand dar.

Ein weiterer Vorteil von Docker ist aber die Reproduzierbarkeit der einzelnen Container. So werden die notwendigen Installationen und Konfigurationen nicht in einem interaktiven System vorgenommen, sondern im sogenannten Dockerfile spezifiziert[3]. Durch dieses Vorgehen erhält man eine Ablaufbeschreibung, welche es vermag, jederzeit absolut idente Container-Images zu erzeugen. Fehleranfällige interaktive Eingaben an der Command-Line werden mit dieser Methodik maßgeblich reduziert.

Beispiel Dockerfile:

```
FROM node:alpine

MAINTAINER Bernd KLAUS "https://berndklaus.at"

RUN apk add --no-cache bash curl g++ gcc libgcc libstdc++ linux-headers make python

USER node

RUN mkdir -p /home/node/app && mkdir -p /home/node/.npm-global

WORKDIR /usr/src/app

ENV USER=node
ENV PATH=/home/node/.npm-global/bin:$PATH
ENV NPM_CONFIG_PREFIX=/home/node/.npm-global

RUN npm install -g loopback-cli && npm cache clean --force

EXPOSE 3000

ENTRYPOINT ["/bin/bash"]
```

Code 1: Dockerfile-Beispiel
Quelle: https://github.com/Berndinox/docker-loopback-cli

Durch den sogenannten Build-Prozess werden die oben angeführten Befehlsabfolgen zu einem Container-Image zusammengeführt. Im konkret angeführten Beispiel wurde eine stark vereinfachte Variante gewählt; der Umfang eines Dockerfile kann je nach Applikation stark an Komplexität und Länge zunehmen. Aufgrund dessen werden viele Container-Images bereits von den Herstellern erzeugt und im Internet via öffentlich zugängliche Container Registries zur Verfügung gestellt.

[2] (Smith, 2019)
[3] (Kumar, 2019)

Ein Container Registry ist ein Verzeichnis, in welchem mehrere Benutzer mittels Dockerfile generierte Images teilen können.

Die Top drei der bekanntesten Registry-Anbieter sind:

- Docker-Hub: https://hub.docker.com/
- Red Hat Quay: https://quay.io/
- Google Container Registry: https://cloud.google.com/container-registry

Für den Aufbau des Prototyps, welcher im zweiten Teil der Arbeit erfolgt, sind alle erforderlichen Docker-Images bereits öffentlich und kostenfrei erhältlich. Welche Container tatsächlich zum Einsatz kommen, wird im späteren Praxisteil genauer erläutert.

1.1.2.2 Aufgaben

Die eigentliche Aufgabe von Kubernetes ist das Managen, Replizieren und kontinuierliche Überprüfen der ausgeführten Container-Instanzen sowie der darunter befindlichen Hostsysteme. Dabei verfolgt Kubernetes einen immutablen sowie deklarativen Ansatz. Ursprünglich erfolgte die Konfiguration eines Systems anhand einer Prozedur, einer Abfolge von inkrementellen einzelnen Schritten wie z. B. „Führe A, B, und C aus“. In einer durch Kubernetes verwalteten Umgebung wird hingegen lediglich wie folgt spezifiziert: „Anzahl Instanzen 3“. Dieses Vorgehen beschreibt den deklarativen Ansatz[4].

Dadurch wird sichergestellt, dass der Zustand nicht nur initial, sondern kontinuierlich gewährleistet wird. Auch können Rollbacks auf vorherige Zustände leichter erfolgen, was bei einem inkrementellen Ansatz (*meist*) nicht der Fall ist. Zusätzlich wird das Hostsystem, auf welchem die Container ausgeführt werden, vollständig und transparent entkoppelt. Der Kubernetes-Scheduler (*siehe 2.2.5*) misst selbstständig die zur Verfügung stehenden Ressourcen und steuert die Zuteilung der Container-Instanzen. Unabhängig davon, ob nur ein oder bis zu 5.000 Hosts[5] für Arbeitslasten zur Verfügung stehen, ist die jeweilige Konfiguration für ein Objekt innerhalb des Kubernetes-Systems ident. Kubernetes abstrahiert die zur Verfügung stehenden Ressourcen und lässt diese als nur eine einzige Einheit erscheinen. So können Anwendungen von kleinen Entwicklungsumgebungen mit nur wenig Aufwand auf größere, produktive Umgebungen übertragen werden.

1.1.2.3 Architektur

Der Aufbau von Kubernetes gliedert sich in zwei Teile: Control Plane für Managementaufgaben sowie Worker Nodes, auf welchen die eigentliche Arbeitslast ausgeführt wird.

Die Control Plane (*siehe Abbildung 4*), sofern Kubernetes als Service bezogen wird, ist vollständig vom Serviceprovider verantwortet und wird von diesem zur Verfügung gestellt. Worker Nodes können on demand zu einer Umgebung hinzugefügt werden. Aufgrund der

[4] (Kelsey Hightower, 2018)

[5] (Kubernetes, 2019)

Komplexität eines eigenständigen Betriebs und der Vielzahl an Serviceangeboten wird in dieser Arbeit auf ein „Kubernetes as a Service“-Modell (*KaaS*) zurückgegriffen. Das bedeutet im Konkreten, dass wir uns vollständig auf die Services innerhalb von Kubernetes konzentrieren können. Im Kapitel „Cloud-Provider“ findet sich eine Auflistung der unterschiedlichen KaaS-Provider.

Obwohl sich diese Arbeit auf ein auf Services basierendes Modell stützt, sei dennoch im Anschluss auf die einzelnen Kubernetes-Komponenten verwiesen. Für das grundlegende Verständnis sowie der Vollständigkeit halber wird kurz auf diese Thematik eingegangen. Dazu ist in Abbildung 4 das Zusammenspiel dargestellt; abschließend werden die einzelnen Komponenten beschrieben.

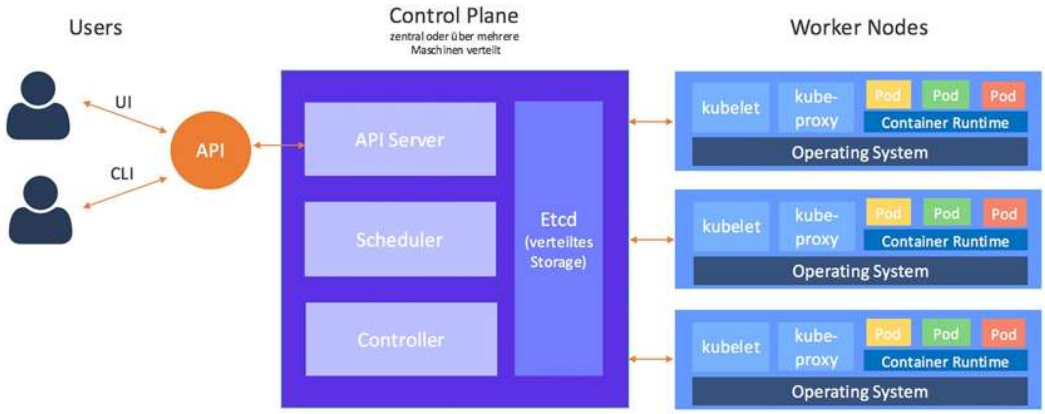

Abbildung 4: Kubernetes-Architektur
Quelle: https://www.scaleuptech.com/

Die Hauptfunktionen der Kubernetes-Komponenten:

- **API-Server**
  Zentraler Zugriffspunkt in Form einer RESTful API, welche für die Steuerung von Kubernetes zuständig ist.

- **Scheduler**
  Ist für die Zuweisung und Überwachung der auszuführenden Container-Instanzen verantwortlich.

- **Controller**
  Überwacht den Zustand und Status von eingespielten Konfigurationen. Die Immutabilität des Systems ist dem Controller zuzuschreiben.

- **etcd**
  Verteilter Key-Value-Datenstore, der als Hauptspeicher für Kubernetes herangezogen wird.

- **Kubelet**
  Primärer Agent auf den Worker Nodes, der zur Kommunikation mit der Control Plane genutzt wird.

- **kube-proxy**
  Agent, welcher interne Services sowie Port Forwards innerhalb des Kubernetes-Clusters verfügbar macht.

Wie bereits erläutert ist bei der Verwendung von KaaS eine detaillierte Kenntnis der jeweiligen Komponenten nicht zwingend erforderlich. Weitere Details entnehmen Sie aktuellen Literaturen.

1.1.2.4 Plug-ins

Plug-ins dienen als Add-ons innerhalb des Kubernetes-Systems. Diese erweitern die Funktionalität von Kubernetes. Insofern Kubernetes als Service angeboten wird, sind derartige Plug-ins durch den Provider vorinstalliert. In Hinsicht auf den Einsatz von WordPress spielen die folgenden beiden Plug-ins eine wichtige Rolle. Zu erwähnen ist, dass die beiden erforderlichen Plug-ins bei allen untersuchten Anbietern verfügbar sind.

- **Netzwerk-Plug-in**
  Erweitert die Funktionalität rund um die Netzwerkschicht in Kubernetes. Durch das Plug-in kann beispielsweise Netzwerkverkehr verschlüsselt werden oder Quality of Service Policies können ausgerollt bzw. Monitoringtools eingebunden werden. In Bezug auf diese Komponente hat sich noch keine Variante als Standard etabliert, weshalb je nach Anbieter unterschiedliche Versionen und Produkte zum Einsatz kommen. Die meist geringen unterschiedlichen Funktionalitäten spielen jedoch für die Bereitstellung von WordPress nur eine untergeordnete Rolle. Eine Ausnahme dabei bildet der Load Balancer (*2.2.3.4 „Ingress“*), welcher für den Zugriff auf WordPress vonnöten ist und dem Bereich „Netzwerk“ zuzuordnen ist. Details zur konkreten Implementierung finden sich in Abschnitt 3.3.2.

- **Storage-Plug-in**
  Plug-in, welches die Funktionen der Speicherschnittstelle erweitert und verbessert. Auch in dieser Rubrik gibt es eine größere Auswahl an zur Verfügung stehenden Plug-ins. Im Unterschied zu den Netzwerk-Plug-ins zeichnet sich hier bereits ein gewisser Standard ab, und zwar das CSI-Plug-in (*Container Storage Interface*). Prinzipielles Ziel der eingesetzten Storage-Plug-ins ist es, innerhalb von Kubernetes eine Schnittstelle zu Speichermedien auf der Cloud-Plattform anzubieten[6]. Die angebotenen Speichermedien reichen von Block Volumes und Fileshares hin bis zu S3-Speichern. Standardmäßig ist die Anbindung nur direkt mittels API oder Weboberfläche des Cloud-Anbieters möglich. Dieses Problem löst das CSI-Plug-in

[6] (Kubernetes, 2019)

und ermöglicht das Generieren der Speicherbereiche direkt aus Kubernetes heraus. Zusätzlich bietet Kubernetes die Möglichkeit, den Lifecycle der erzeugten Volumes automatisch zu verwalten. Dieses Plug-in spielt eine wichtige Rolle für den Einsatz von WordPress innerhalb von Kubernetes, um Daten nachhaltig zu sichern. Der Einsatz von PersistentVolumes (*siehe 2.2.3.4 „Objekte“*), also nicht flüchtigem Speicher, ist für den späteren Prototyp unerlässlich.

1.1.3 Einsatz

In diesem Kapitel wird der eigentliche Einsatz von Kubernetes beschrieben. Alle weiter angeführten Abschnitte zielen auf die Bedienung des Kubernetes-Systems selbst ab.

1.1.3.1 API

Ausgangpunkt für alle Operationen, welche in Kubernetes ausgeführt werden können, ist die API-Schnittstelle. Diese ist via HTTPS-Protokoll erreichbar und bildet sämtliche Objekte des Kubernetes-Clusters als sogenannte REST-Objekte ab.

In Abbildung 4 ist die API visuell als Komponente zwischen den Benutzertools sowie den Kubernetes-Kernsystemen dargestellt. Tools wie kubectl oder das Kubernetes-Dashboard greifen im Hintergrund auf die API-Schnittstelle von Kubernetes zu und machen die von der API erzeugten Objekte für den Menschen leichter lesbar bzw. können Objekte in vereinfachter Form bearbeitet werden.

1.1.3.2 kubectl

Wie im vorherigen Abschnitt bereits angedeutet ist kubectl das Standard-CLI-Tool (*Command-Line Interface*) zum Interagieren mit der Kubernetes-API. Mittels kubectl können Objekte erstellt, bearbeitet sowie entfernt werden. Im Weiteren bietet es die Möglichkeit, Cluster-, Troubleshooting- sowie Debugging-Operationen auszuführen. kubectl ist für alle gängigen Betriebssysteme erhältlich. Ein schlichtes „kubectl“ an der CLI gibt die Hilfeseite aus (*siehe Anhang, „Kubectl“*), anhand welcher man sich sehr gut orientieren kann. Die Konfiguration von kubectl wird von den Serviceprovidern bereitgestellt und ist am jeweiligen Client unter „`$PWD/.kube/config`“ zu hinterlegen. Die Konfiguration enthält den Zugriffspunkt der API sowie Informationen zur Authentifizierung im Cluster. Resümierend lässt sich zusammenfassen, dass kubectl das Standardwerkzeug für Administratoren zum Bedienen eines Kubernetes-Clusters darstellt.

1.1.3.3 YAML

YAML (*kurz für Yet Another Markup Language*) dient der Beschreibung von Objekten innerhalb der Kubernetes-Welt. Grundsätzlich können alle Objekte und deren Zustände durch kubectl deklariert werden; dies führt jedoch schnell dazu, dass Befehle lang und unübersichtlich werden. YAML bietet eine für den Menschen einfach lesbare Beschreibungssprache[7]. Unter Einsatz von YAML-Konfigurationsdateien können auch die komplexesten Szenarien übersichtlich gestaltet bzw. beschrieben werden. Für den

[7] (Oren Ben-Kiki, 2009)

produktiven Einsatz sind dementsprechend Konfigurationsdateien einer interaktiven Eingabe mittels kubectl stets vorzuziehen.

Im Anschluss findet sich ein kurzer Ausschnitt einer YAML-Datei, um deren Charakteristika zu skizzieren.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: demo
  labels:
    app: demo
spec:
  replicas: 1
  selector:
    matchLabels:
      app: demo
    spec:
      containers:
        - image: busybox
          name: demo-app
  restartPolicy: Always
```

Code 2: Deployment

Der Aufbau selbst ist für nicht technikaffine Personen schnell nachvollziehbar. Ein Unterobjekt ist jeweils zwei Abstände nach innen gerückt gegenüber dem darüber befindlichen Elternobjekt. Auf die Elemente, wie sie im oben ersichtlichen Beispiel angeführt sind, wird im Weiteren in den anschließenden Kapiteln noch detaillierter eingegangen. Als Alternative zu YAML steht im Weiteren auch JSON (*JavaScript Object Notation*) zur Verfügung. Da YAML jedoch das einfachere Format bietet, findet JSON weitaus weniger oft Verwendung[8].

1.1.3.4 Objekte

In diesem Kapitel werden die einzelnen Objekte innerhalb des Kubernetes-Clusters mittels des Einsatzes der zuvor beschriebenen Tools und Beschreibungssprachen näher erörtert. Kubernetes bietet eine Vielzahl an unterschiedlichen Objekttypen, welche im späteren Prototyp ihren Einsatz finden. In der folgenden Abbildung (*siehe Abbildung 5*) werden die wichtigsten und elementaren Objekte dargestellt. Dabei wird ein Top-down-Ansatz verfolgt. Vom Benutzer aus werden abwärts die miteinander in Verbindung stehenden Objekttypen skizziert. Ebenso wird dargestellt, in welcher Sphäre sich das jeweilige Objekt befindet. Im Anschluss daran werden die in der Zeichnung verwendeten Objekte beschrieben und es wird auf deren Funktion eingegangen. Vorweg die wichtigsten Objekte: Ingress, Service, Deployment, ReplicaSet, Pod, StorageClass, PersistentVolumeClaims, Volume.

[8] (Kubernetes, 2020)

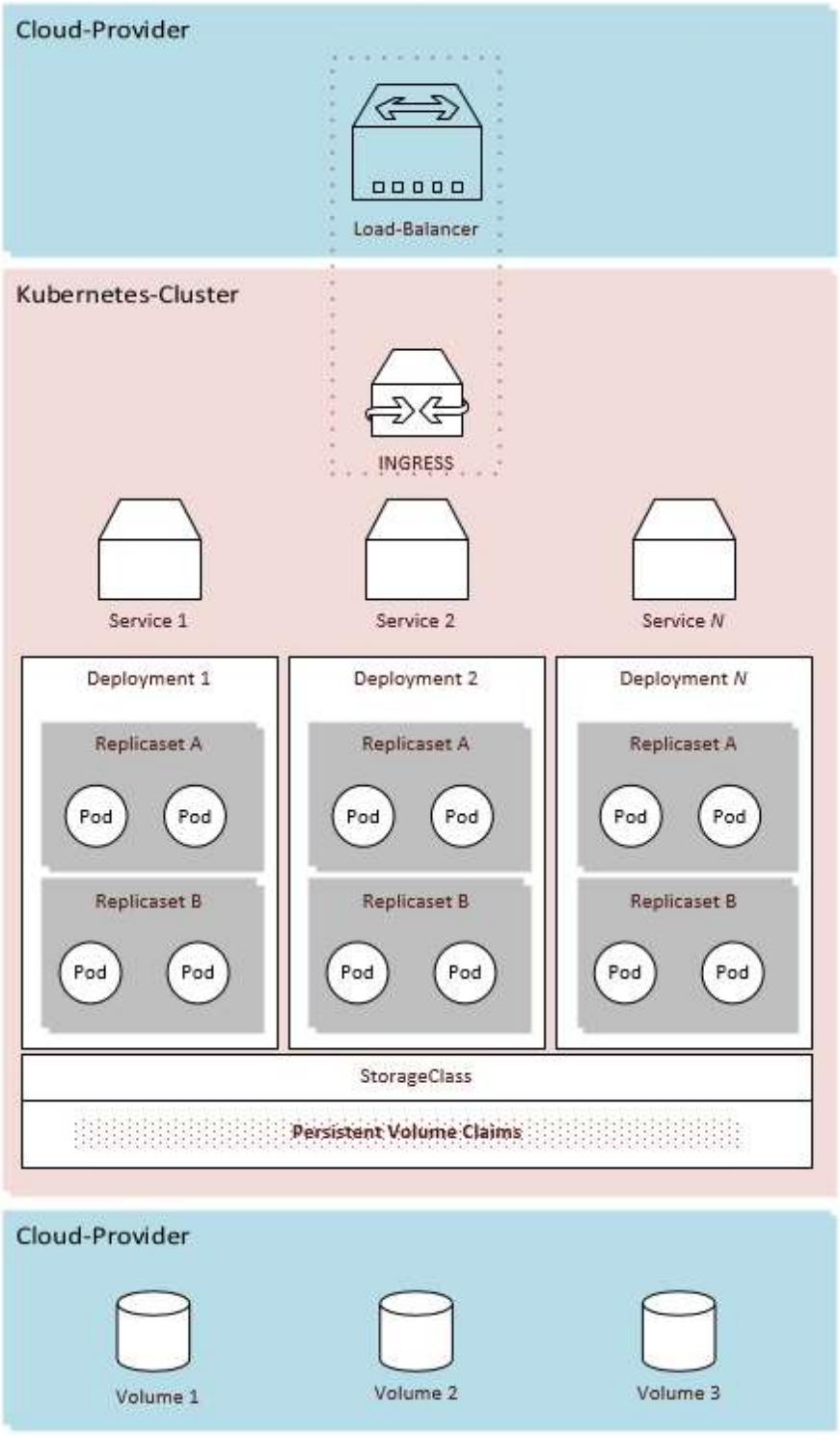

Abbildung 5: Kubernetes-Objekte

Die im roten Bereich befindlichen Elemente befinden sich innerhalb des Kubernetes-Systems; hingegen sind Objekte, welche sich am bläulichen Hintergrund befinden, eine Ressource auf der Cloud-Plattform selbst.

Ingress

Stellt eine Ressource dar, welche auf OSI-Layer 7 als HTTP(S) Load Balancer fungiert[9]. Standardmäßig ist in einem Kubernetes-Cluster noch kein Ingress-Controller vorhanden, weshalb dieser erst erzeugt werden muss. Zur Auswahl stehen Ingress-Controller von diversen Herstellern. Der Standard-Controller, welcher auch am häufigsten zum Einsatz kommt, ist ingress-nginx[10]. Dieser wird ständig weiterentwickelt; des Weiteren hat sich bereits eine große Community rund um das Projekt gebildet. Der Controller ist als Open-Source-Software lizenziert und somit auch frei erhältlich und einsetzbar. Eine Einheit an Controller-Instanzen ist dabei jeweils einem dedizierten Load-Balancer zugewiesen. Der zugehörige Load-Balancer wird bei der Installation des Ingress-Controllers automatisch vom Cloud-Provider generiert sowie konfiguriert. In der oben angeführten Grafik sind die beiden Objekte deshalb symbolisch durch punktierte Linien verbunden. Durch den erzeugten Load-Balancer wird der Ingress-Controller über das World Wide Web adressiert. Im Vergleich zur herkömmlichen Umgebung bildet der Ingress-Controller den Reverse Proxy ab (*siehe Abbildung 2*).

Nachfolgend werden die zur Bereitstellung einer WordPress-Installation wichtigsten Features des Ingress-Controllers angeführt.

- **Server Name Identification**

  Ermöglicht die Host-based-Routing-Funktionalität. Das bedeutet im Weiteren, dass hinter einer Load-Balancer-IP-Adresse mehrere unterschiedliche Domains betrieben werden können. Der Ingress-Controller übernimmt dabei die Rolle einer Routing-Engine.

- **Proxy Protocol**

  Um auch im Back-end die realen IP-Adressen der Besucher verarbeiten zu können, ist der Einsatz eines Proxy Protocol unumgänglich. Dabei schreibt der Load-Balancer die IP des Besuchers in den Header der Webanfrage. Der Ingress-Controller kann die IP-Adresse so auslesen und an das Backend weitergeben. In diesem Vorgang wird das HTTP-Header-Feld „X-Fowarded-For“ verwendet. Ohne Proxy Protocol würde am Backend lediglich die IP-Adresse der internen Subnet-IP des Load-Balancers aufscheinen.

- **SSL**

  Die Verwendung von Secure Socket Layer Protocol ist heutzutage bereits ein absolutes Muss. Moderne Internetbrowser kennzeichnen bereits Websites explizit, welche nicht über eine verschlüsselte Verbindung aufgerufen werden können. Ingress-Nginx unterstützt nativ den Einsatz von Zertifikaten, um eine HTTP-Verbindung mittels SSL abzusichern. Auch lässt sich durch die Erweiterung Cert-

[9] (Ingress-Nginx Docs, 2019)

[10] (Ingress-Nginx GitHub, 2019)

Manager eine nahtlose Integration des Service Let's Encrypt[11] erreichen. Durch die Nutzung des Service können ansonst kostenpflichtige Zertifikate gratis und automatisiert in das System eingespielt werden. Dies bringt für stark automatisierte Umgebungen einen deutlichen Vorteil gegenüber einer manuellen Pflege der Zertifikate.

Service

Ein Service referenziert jeweils eine Gruppe von identen Pods bzw. Container-Instanzen und dient als Kubernetes-interner Load-Balancer für diese Ziele. Dabei wird die Anzahl der Pods durch den Kubernetes-Cluster überwacht und automatisch dem Service hinzugefügt oder auch entfernt. Durch diesen Mechanismus ist sichergestellt, dass nur „gesunde" Pods über ein Serviceobjekt erreichbar sind. Dies erhöht die Zuverlässigkeit des Systems signifikant und garantiert eine sehr hohe Erreichbarkeitsquote. Ein weiteres wichtiges Merkmal, welches dadurch erreicht wird, ist jenes der Skalierbarkeit. Werden idente Pods vervielfältigt, so kümmert sich Kubernetes automatisch darum, diese dem Serviceobjekt zuzuordnen und so verfügbar zu machen. Arbeitslasten können so vollautomatisch über mehrere Instanzen hinweg verteilt werden.

Deployment

Dient dem Management einer Softwareversion innerhalb von Pods respektive deren Replica-Sets. Die beiden letzteren Objekte werden im Anschluss beschrieben. Ein Deployment bietet zur Verwaltung von Softwareversionen Mechanismen, mit welchen rollierende Updates durchgeführt sowie überwacht werden können. In Abschnitt 2.2.3.3 ist bereits beispielhaft eine minimale Ausprägung eines Deployment angeführt. Für den in dieser Arbeit untersuchten Sachverhalt spielen Deployments eine wesentliche Rolle. WordPress erscheint regelmäßig in neuen Versionen. Durch die Anwendung von Deployments kann die Erreichbarkeit von WordPress während des Upgrade-Vorgangs sichergestellt werden. Diese Funktionalität bietet einen enormen Mehrwert im Gegensatz zu einer herkömmlichen Bereitstellung in einer virtuellen Maschine, da die Serviceverfügbarkeit um ein Vielfaches gesteigert werden kann.

Replica-Set

Dient als Vorlage bzw. Template für eine Anzahl *N* an Pods. Dabei werden die Eigenschaften des Replica-Sets auf die untergeordneten Pods vererbt. Das Hauptaufgabengebiet eines Replica-Sets ist das Skalieren von Pods. Dabei wird der Replizierungsprozess nicht einmalig ausgeführt, beispielsweise um von einem auf zwei Pods zu skalieren, sondern vielmehr erfolgt eine ständige Überwachung der Pod-Anzahl. Reagiert ein Pod nicht mehr oder wird dieser durch ein Update ersetzt, so stellt das Replica-Set sicher, dass immer die gewünschte Anzahl an Pods im System ausgeführt wird. Dieses

[11] (LetsEncrypt, 27)

Objekt stellt somit einen wichtigen Baustein zur Verfügung, um die Immutabilität[12] (*siehe 2.2.2.2, „Aufgaben“*) innerhalb des Kubernetes-Clusters zu gewährleisten. Wie in Abbildung 5 ersichtlich werden Replica-Sets von Deployments gesteuert, Pods hingegen von Replica-Sets. So ergibt sich eine logische Verkettung vonElementen.

Pods

Kleinste Einheit, welche innerhalb von Kubernetes generiert werden kann. Ein Pod verkörpert eine ausführbare Instanz. In einem Pod können jedoch auch mehrere unterschiedliche Docker-Container (*siehe 2.2.2.1*) parallel laufen. In den meisten Fällen, auch bei unserem konkreten Anwendungsfall von WordPress, kommt jedoch nur ein Container zum Einsatz. Alle Docker-Container in einem Pod teilen sich die lokalen Ports sowie Storage Volumes. Ein denkbares Einsatzszenario, in welchem mehrere Container innerhalb eines Pods ausgeführt werden, wäre der Einsatz von Agents, welche die Hauptanwendung unterstützen. Man spricht dann von sogenannten Sidecars[13]. Das Starten eines einzelnen Pods innerhalb eines Kubernetes-Clusters ist prinzipiell möglich, entspricht jedoch einem Anti-Pattern, da auf viele Hilfsmittel, wie sie durch Kubernetes bereitgestellt werden, verzichtet wird. So ist ein Pod meist ein Endprodukt, welches zwingend erforderlich ist, aber nicht direkt selbst erstellt wird. Vielmehr wird durch übergeordnete Elemente, wie ein Deployment, StatefulSet oder DaemonSet, die Erzeugung von Pods angestoßen und überwacht.

Namespaces

Ermöglichen es, eine logische Trennung zwischen Objektgruppen herzustellen. So können einzelne Anwendungen, aber auch Entwicklungs- sowie Produktivumgebungen voneinander getrennt werden. Aber nicht nur die logische Trennung wird mit Namespaces erreicht, im Weiteren können den gruppierten Objekten auf Namespace-Ebene Ressourcenlimits zugewiesen werden. Ebenfalls kann granular gesteuert werden, welcher Administrator Zugriff auf welchen Namespace erhält; so können Cluster-Ressourcen sicher an unterschiedliche Teams oder Personen aufgeteilt werden. Im Prototyp werden alle zu einer WordPress-Seite gehörigen Objekte in einem dedizierten Namespace hinterlegt. Damit wird eine logische Trennung der Kunden erreicht; zusätzlich können auf Namespace-Ebene Metriken zur Ressourcenausnutzung gemessen werden. Ebenso erwähnt werden muss, dass durch eine derartige Kategorisierung enorm an Übersichtlichkeit gewonnen wird. In komplexen Umgebungen können damit Fehlerquellen verringert werden.

[12] (Kelsey Hightower, 2018)

[13] (Bilgin Ibryam, 2019)

ConfigMaps und Secrets

Die Einsatzweise sowie die Funktionalität von ConfigMaps und Secrets sind nahezu ident, weshalb diese Items in nur einem Abschnitt beschrieben werden können. Prinzipielles Ziel der beiden Objekte ist es, die Möglichkeit zu bieten, einem Pod bzw. dessen Container-Instanz Informationen zukommen zu lassen. Die beiden Items bestehen dabei jeweils aus einem Schlüsselwert- und Inhaltspaar. Der Inhalt des Datenfelds der beiden Objekte führt zur eigentlichen Unterscheidung. ConfigMaps enthalten unverschlüsselte Strings oder auch ganze Dateien; hingegen werden in Secrets Kennwörter oder Zertifikate in verschlüsselter Form verfügbar gemacht.

Nach Erstellung können die Schlüsselwerte jeweils aus einem Deployment, Daemon-Set oder auch Stateful-Set heraus referenziert werden. Diese Verlinkung ermöglicht es, die Konfiguration eines Deployment völlig ident zu halten, während via ConfigMap oder Secret unterschiedliche Informationen an die Pods propagiert werden können. So kann beispielsweise einfach zwischen unterschiedlichen Kennwörtern gewechselt werden. Auch können die Applikationen in den Containern mit unterschiedlichen Konfigurationen ausgestattet werden und so dynamisch adaptiert werden.

Nachfolgend ein kurzes Beispiel eines Secret. Als erster Schritt muss das Passwort in einen Base64-dekodierten String umgewandelt werden.

```
echo -n 'password' | base64
cGFzc3dvcmQ=
```

Code 3: Passwort-Erzeugung

Anschließend wird ein YAML-File erzeugt, welches den Schlüsselwert und den dazu passenden Inhalt repräsentiert.

```
apiVersion: v1
kind: Secret
metadata:
  name: secret-01
type: Opaque
data:
  password: cGFzc3dvcmQ=
```

Code 4: Secret

In einem Deployment kann nun durch Referenzieren auf das Secret „secret-01“ und den enthaltenen Schlüsselwert „password“ das zuvor generierte Kennwort in den Pod geladen werden.

StorageClass

Steht für einen auf der Cloud-Plattform definierten Speicherbereich, welcher für Kubernetes reserviert wurde. Um Volumes erzeugen zu können, siehe dazu auch „Persistent Volume“, ist eine StorageClass eine notwendige Grundvoraussetzung. Die Speicherbereiche können sich in ihrer Geschwindigkeit sowie deren Funktionsumfang unterscheiden. Eine der wichtigsten funktionellen Unterscheidungen ist jene, in welchem Modus auf das Medium zugegriffen werden kann.

Man unterscheidet:

- ReadWriteOnce: Lese- und Schreibzugriff für nur einen Pod
- ReadOnlyMany: lesender Zugriff für mehrere Pods
- ReadWriteMany: Lese- und Schreibzugriff für mehrere Pods

Um eine konsistente Skalierbarkeit der Anwendungen zu gewährleisten, ist eine der beiden letzteren Varianten Voraussetzung. In der Ausprägung „ReadWriteOnce“ wird im Fall der Skalierung lediglich ein neues leeres Volume erzeugt und den Pods zugewiesen.

Persistent Volume

Das grundsätzliche Einsatzgebiet von Kubernetes ist das Orchestrieren von zustandslosen Container-Instanzen. Das bedeutet, dass Daten, welche durch Benutzerinteraktion während der Laufzeit in eine Instanz eingegeben werden, nur innerhalb der ausführenden Instanz bekannt sind. Wird die Instanz beendet, was durch die Steuerung von Replica-Sets regelmäßig der Fall sein kann, sind die eingegebenen Daten verloren. Dieses Verhalten leitet sich davon ab, dass Kubernetes ursprünglich für Systeme entwickelt wurde, um verteilte Berechnungen in einer großen Anzahl an Instanzen durchzuführen und diese anschließend wieder zusammenzuführen und in anderen Systemen weiterzuverarbeiten oder zu sichern. In diesem Szenario war eine persistente Zwischenspeicherung nicht erforderlich. Im Laufe der Zeit wurde jedoch erkannt, dass für eine große Anzahl an Anwendungen ein nachhaltig gesicherter Zustand der Daten absolut unerlässlich ist. Im Fachjargon unterscheidet man zwischen „stateless“ und „stateful“. Wie bereits in Abschnitt 2.2.2.4 erwähnt kommt zur Verwaltung von persistenten Speichern CSI zum Einsatz. Jedes Persistent Volume ist an eine StorageClass gebunden. Es steht für einen eindeutig definierten Speicherbereich. Jedes Volume kann weiters mit Optionen wie Größe oder verfügbarer Speichergeschwindigkeit (*QoS*) versehen werden.

Im Prototyp werden PersistentVolumeClaims eingesetzt. Diese setzen im Hintergrund ebenso auf die Verwendung von Persistent Volumes und StorageClasses, bieten jedoch einen Mechanismus, um die notwendigen Volumes, welche üblicherweise manuell erstellt werden müssen, automatisiert im Bedarfsfall zu erzeugen. Durch diese Methode können manuelle Eingriffe minimiert werden. Man definiert lediglich die Basisoptionen des Volume, Kubernetes kümmert sich anschließend um den vollständigen Lebenszyklus der jeweiligen Volumes.

Weitere Objekte

Wie bereits erwähnt bietet Kubernetes noch weitere Objekte wie das DaemonSet oder Stateful-Set, welche einem Deployment sehr ähneln. Ebenso besteht die Möglichkeit, CronJobs oder OneShots einzusetzen. All diese Objekte sind für die vorliegende Arbeit

jedoch nicht von Relevanz, weshalb diese Objekte nicht weiter beschrieben werden. Der Vollständigkeit halber sollten diese jedoch Erwähnung finden.

1.2 Cloud-Provider

Das Nutzen von Services in der Cloud bietet einige Vorteilen[14] im Gegensatz zu einem Betrieb von IT in einem lokalen Rechenzentrum. Die Hauptvorteile sind in folgender Aufzählung angeführt:

- Höhere Skalierbarkeit
- On-demand-Provisionierung
- Delegation von Verantwortung
- Höhere Verfügbarkeit
- Verbesserte Servicequalität

Zu beachten jedoch ist, dass durch den Einsatz von Cloud-Technologie eine höhere Abhängigkeit entsteht. Dementsprechend wird an dieser Stelle explizit darauf hingewiesen, Prozesse, welche das Kerngeschäft betreffen sowie einen Wettbewerbsvorteil gegenüber der Konkurrenz bieten, nicht oder nur nach entsprechender Abwägung in die Cloud zu migrieren.

1.2.1 Public Cloud

Gegenstand dieser Arbeit ist, wie im einführenden Teil beschrieben, die Untersuchung der Kosten für den privaten Gebrauch. Dementsprechend geeignet ist das Servicemodell einer Public Cloud, welche den Einstieg bereits mit geringen Mitteln ermöglicht. Hingegen sind die Modelle Private Cloud sowie Hybrid Cloud aufgrund der damit einhergehenden höheren Kosten nicht geeignet und finden in der weiteren Betrachtung keine Berücksichtigung.

In der folgenden Grafik wird die prozentuelle Nutzung der größten Cloud-Anbieter verglichen.

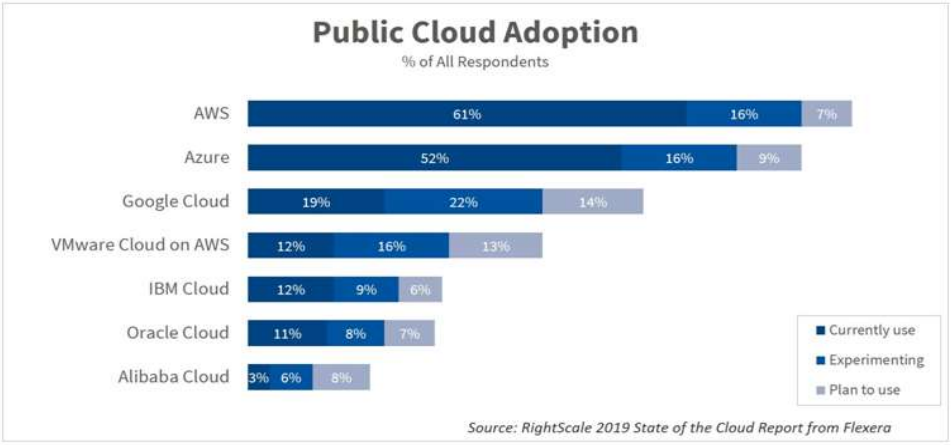

Abbildung 6: Public-Cloud-Anteile
Quelle: zdnet.com

[14] (Alexander Redlein)

Aufgrund des Umfangs wird der Fokus auf die Top drei der Anbieter gelegt. Im Praxisteil der Arbeit kommt es demzufolge zu einer Gegenüberstellung zwischen den folgenden Cloud-Providern:

- Amazon Web Services
- Microsoft Azure
- Google Cloud

Weitere Alternativen, welche ebenfalls Kubernetes als Service anbieten, wären OVH.de, Scaleway.com oder auch DigitalOcean.com. Jedoch befindet sich bei diesen Providern das Service noch in einem experimentellen Zustand (*Stand: 17.11.2019*) und sie sind deshalb für den produktiven Einsatz weniger geeignet. Hingegen ist der Einsatz von Kubernetes as a Service der drei eingangs ausgewählten Anbieter bereits vielfach erfolgserprobt und bietet demnach eine gute Servicequalität und Verfügbarkeit. Insbesondere bei der Nutzung eines Service aus der Cloud ist die Stabilität ein absolutes Muss-Kriterium. Jedenfalls ist damit zu rechnen, dass in Zukunft noch weitere Anbieter Kubernetes as a Service in erforderlicher Güte bzw. Qualität anbieten werden.

1.2.2 Provider-Übersicht

Im Weiteren werden die Eckdaten, Informationen zum gewählten Produkt sowie eine grobe Aufstellung der Kosten zu den gewählten Providern kategorisch dargestellt. Betreffend die in den folgenden Kapiteln dargestellten Kosten ist anzuführen, dass sich sämtliche Kosten auf ein Pay-per-Use-Prinzip beziehen. Weiters wird eine Lokation in Europa gewählt. Eine genaue Aufstellung der Kosten sowie den tatsächlich entstehenden Bedarf entnehmen Sie der Kostenrechnung im zweiten Teil der vorliegenden Arbeit.

1.2.2.1 Amazon Web Services

Der de facto größte Cloud-Anbieter weltweit. 2006 wurde Amazon Web Services (*AWS*) als Tochterunternehmen von Amazon.com gegründet. Die ursprüngliche Idee war es, Ressourcen, welche für den eigenen Einsatz temporär nicht benötigt wurden, anderen Unternehmen zur Verfügung zu stellen. AWS gilt als Vorreiter im Cloud-Sektor und bietet eine große Anzahl an unterschiedlichen Produkten an, welche vom einfachen virtuellen Server bis hin zu Produkten, welche durch den Einsatz von künstlicher Intelligenz Daten analysieren können, reichen.

In unserem konkreten Fall sind folgende Produkte von Relevanz:

- Elastic Kubernetes Service (*EKS*)
- Elastic Compute Cloud (*EC2*)
- Elastic Load Balancing (*ELB*)
- Elastic Block Storage (*EBS*)
- Elastic File System (*EFS*)

Kosten

Die Gesamtkosten summieren sich aus dem jeweiligen Bedarf der tatsächlich verwendeten Services auf Stundenbasis und setzen sich wie folgt zusammen[15].

Tabelle 1: AWS-Preise

Produkt	Variante	Preis in USD	Einheit
EKS		0.20	Stunde
EC2	variabel	0.006 bis 30.584	Stunde
ELB		0.027	Stunde
EBS	HDD SSD	0.054 0.119	GB GB
EFS	Standard Cold	0.36 0.025	GB GB

1.2.2.2 Microsoft Azure

Bereits 1975 wurde in New Mexico Microsoft mit dem Ziel, Computer auch der breiten Masse zugänglich zu machen, gegründet. Dies gelang in weiterer Folge auch. Seither konnte Microsoft den IT-Sektor mit zahlreichen innovativen Entwicklungen und Produkten vorantreiben und gestalten. 2010 wurde Microsoft Azure in seiner ersten Version veröffentlicht. Seither wurde die von Microsoft betriebene Cloud-Plattform ständig weiterentwickelt und verbessert, sodass heute eine Vielfalt an unterschiedlichen Services verfügbar ist. Auch konnte Microsoft Azure in den letzten Jahren gut an den Erfolg von AWS anschließen und sich an die zweite Stelle der Public-Cloud-Anbieter einreihen (*siehe Abbildung 6*). Vor allem am europäischen Markt konnte Microsoft im Vergleich zu AWS ein größeres Wachstum verzeichnen.

Folgende Services sind für die Umsetzung auf Microsoft Azure erforderlich:

- Azure Kubernetes Service (*AKS*)
- Azure Virtual Machines (VMs)
- Azure Load Balancer (*ALB*)
- Azure Managed Disks (*AMDs*)
- Azure File Storage (*Azure Files*)
- Azure App Service (*AAS*)

[15] (Amazon Web Services, 2020)

Kosten

Die Kosten errechnen sich ebenfalls aus dem tatsächlich entstandenen Bedarf auf Stundenbasis. Zu beachten ist, dass der angeführte Preis des Load-Balancers nur für die ersten fünf Regeln gültig ist. Im später erfolgenden Vergleich werden jedoch lediglich zwei Regeln benötigt, weshalb weitere Details zum Preis nicht angeführt sind.

Tabelle 2: Azure-Preise

Produkt	Variante	Preis in EUR	Einheit
AKS		Frei nutzbar	
VMs	variabel	0.038 bis 8.312	Stunde
ALB	Nach Regeln	0.022	Stunde
AMD	SSD Premium SSD Standard HDD Standard	ab 0.28 ab 0.13 ab 1.30	4 GB 4 GB 32 GB
Azure Files[1]	Premium Standard	0.223 0.051	GB GB
AAS	Basic Standard Premium	ab 0.064€ ab 0.085€ ab 0.169€	Stunde

[1] Zusätzlich werden ausgeführte Operationen wie List, Put, Create mit 0,0127 € pro 10.000 Vorgängen berechnet.

1.2.2.3 Google Cloud

Meist wird mit dem Begriff „Google“ die Suchmaschine verbunden. Aber Google ist nicht nur ein Suchmaschinenanbieter, sondern betreibt bzw. bietet eine große Anzahl an weiteren unterschiedlichen Produkten wie etwa Google Maps, YouTube, Google Chome, Android, Gmail, Google Drive, Google Ads und auch Kubernetes an. Eine vollständige Liste findet sich auf der Homepage[16] von Google. Für die vorliegende Arbeit von Relevanz ist die Google-Cloud-Plattform, welche seit 2011 am Markt verfügbar ist. Positiv besticht, dass Google als erster Provider Kubernetes as a Service in seinem Portfolio anbieten konnte. Wie bereits in Kapitel 2.2.1 erwähnt wurde, ist Google auch der ursprüngliche Entwickler von Kubernetes, weshalb diesbezüglich enormes Know-how in den Reihen von Google vorherrscht. Durch dieses Wissen konnte die erste nahtlose Integration von Kubernetes in eine Cloud-Plattform erreicht werden.

Die folgenden Elemente der Google-Cloud-Plattform finden für unseren Prototyp Verwendung.

[16] (Google LLC, 15)

- Google Kubernetes Engine (*GKE*)
- Google Compute Engine (*GCE*)
- Cloud Load Balancing (*GLB*)
- Persistent Disk (*GPD*)
- Cloud Filestore (*Filestore*)

Kosten

Wie auch bei den zuvor angeführten alternativen Anbieter erfolgt eine bedarfsgerechte Berechnung der Kosten auf Stundenbasis. Zu beachten ist, dass der angeführte Preis des Load-Balancers nur für die ersten fünf Regeln gültig ist.

Tabelle 3: Google-Preise

Produkt	Variante	Preis in USD	Einheit
GKE		Frei nutzbar	
GCE	variabel	0.0612 bis 13.092	Stunde
GLB	Nach Regeln	0.025	Stunde
GPD	Standard SSD	0.048 0.204	GB/Monat GB/Monat
Filestore[1]	Premium Standard	0.000411 0.000274	GB/Stunde GB/Stunde

[1] Bei der Variante Standard ist mindestens 1 Terrabyte und bei der Variante Premium sind 2,5 Terrabyte zu beziehen.

1.2.3 Preisgestaltung

Betreffend die in Kapitel 2.3.2 angeführten Preise ist festzuhalten, dass es sich bei den angeführten Werten lediglich um eine Momentaufnahme handelt. Die Preise können sich je nach Region, Vertragsgestaltung oder durch Adaptionen seitens des Anbieters entsprechend verändern.

Literaturverzeichnis

Alexander Redlein, P. V. (kein Datum). *Organisation und Betrieb von IT-Abteilungen.* Wiener Neustadt: Ferdinand Porsche FernFH.

Amazon Web Services. (14. 01 2020). *AWS Pricing EC2*. Von https://aws.amazon.com/de/ec2/pricing/on-demand/ abgerufen

Amazon Web Services Price. (19. 04 2020). *AWS EKS Preise*. Von AWS EKS Preise: https://aws.amazon.com/de/eks/pricing/ abgerufen

Bilgin Ibryam, R. H. (2019). *Kubernetes Pattern.* O'Reilley.

Bitnami. (12. 04 2020). *Bitnami*. Von https://bitnami.com/ abgerufen

Cert-Manager Authors. (12. 04 2020). *Cert-Manager*. Von https://cert-manager.io/docs/ abgerufen

Diamanti. (2018). *Container Adoption Benchmark Servey.* Diamanti.

GitHub. (10. September 2019). Von https://github.com/WordPress/WordPress abgerufen

gnu.org. (25. Oktober 2017). *gnu.org*. Von https://www.gnu.org/licenses/old-licenses/gpl-2.0.html abgerufen

Google LLC. (15. 01 2020). *Google Produktübersicht*. Von https://about.google/intl/de/products/ abgerufen

Goolge-Cloud-Plattform. (19. 04 2020). *Google Cloud FileStore Pricing*. Von https://cloud.google.com/filestore/pricing abgerufen

Hetzl, A. (2012). *WordPress 3 – Das umfassende Handbuch.* Galileo Press.

Ingress-Nginx Docs. (14. 11 2019). Von https://kubernetes.github.io/ingress-nginx/ abgerufen

Ingress-Nginx GitHub. (14. 11 2019). Von https://github.com/kubernetes/ingress-nginx/ abgerufen

Kelsey Hightower, B. B. (2018). *Kubernetes – Eine kompakte Einführung.* Heidelberg: dpunkt.verlag.

Kernl.us. (21. 04 2020). *Kernl.us Blog*. Von https://blog.kernl.us/2019/03/is-more-ram-or-more-cpu-better-for-wordpress-hosting/ abgerufen

Kubernetes. (09. 11 2019). *Kubernetes*. Von https://kubernetes.io/docs/setup/best-practices/cluster-large/ abgerufen

Kubernetes. (10. 11 2019). *Kubernetes Blog*. Von https://kubernetes.io/blog/2019/01/15/container-storage-interface-ga/ abgerufen

Kubernetes. (14. 01 2020). *Kubernetes Concepts*. Von https://kubernetes.io/docs/concepts/configuration/overview/ abgerufen

Kumar, A. (2019). *Docker & Kubernetes Fundamentals.*

LetsEncrypt. (27. 01 2020). Von https://letsencrypt.org/ abgerufen

McKeown, S. (2019). On forking WordPress, Forks in general, early WordPress and the community. In *Chapter 3.*

Microsoft LLC. (12. 04 2020). *Linux Virtual Machines Pricing*. Von https://azure.microsoft.com/en-us/pricing/details/virtual-machines/linux/ abgerufen

Microsoft LLC. (11. 04 2020). *Microsoft App Service Pricing*. Von https://azure.microsoft.com/en-us/pricing/details/app-service/windows/ abgerufen

Microsoft LLC. (12. 04 2020). *MS Docs: Azure Files Dynamic PVC*. Von https://docs.microsoft.com/en-us/azure/aks/azure-files-dynamic-pv abgerufen

Moore, J. D. (2019). *Kubernetes – The Complete Guide to Master Kubernetes.* Amazon.

Netcraft Company. (07. 11 2019). *Netcraft*. Von https://news.netcraft.com/archives/2019/02/28/february-2019-web-server-survey.html abgerufen

Oren Ben-Kiki, C. E. (01. 10 2009). *YAML Ain't Markup Language.* Von https://yaml.org/spec/1.2/spec.pdf abgerufen

PHP. (07. 11 2019). *PHP.Net Library*. Von https://wiki.php.net/internals/windows/stepbystepbuild abgerufen

Schäferhoff, N. (2018). *Popular CMS by Market Share.* websitesetup.

Smith, B. (2019). *Docker – A Step by Step Guide to Learn and Master Docker.* SJ Publishing.

Wikipedia. (27. 01 2020). *Wikipedia*. Von https://de.wikipedia.org/wiki/.htaccess abgerufen

WordPress. (10. September 2019). *Wordpress.org*. Von https://de.wordpress.org/about/requirements/ abgerufen

WordPress. (10. September 2019). *Wordpress.org*. Von https://de.wordpress.org/category/release/ abgerufen

Anhang

Inhaltsverzeichnis

KUBECTL 51

kubectl

```
kubectl controls the Kubernetes cluster manager.
 Find more information at: https://kubernetes.io/docs/reference/kubectl/overview/
Basic Commands (Beginner):
  create         Create a resource from a file or from stdin.
  expose           Take a replication controller, service, deployment or pod and expose it as a new
Kubernetes Service
  run            Run a particular image on the cluster
  set            Set specific features on objects
Basic Commands (Intermediate):
  explain        Documentation of resources
  get            Display one or many resources
  edit           Edit a resource on the server
  delete          Delete resources by filenames, stdin, resources and names, or by resources and label
selector
Deploy Commands:
  rollout        Manage the rollout of a resource
  scale          Set a new size for a Deployment, ReplicaSet, Replication Controller, or Job
  autoscale      Auto-scale a Deployment, ReplicaSet, or ReplicationController
Cluster Management Commands:
  certificate    Modify certificate resources.
  cluster-info   Display cluster info
  top            Display Resource (CPU/Memory/Storage) usage.
  cordon         Mark node as unschedulable
  uncordon       Mark node as schedulable
  drain          Drain node in preparation for maintenance
  taint          Update the taints on one or more nodes
Troubleshooting and Debugging Commands:
  describe       Show details of a specific resource or group of resources
  logs           Print the logs for a container in a pod
  attach         Attach to a running container
  exec           Execute a command in a container
  port-forward   Forward one or more local ports to a pod
  proxy          Run a proxy to the Kubernetes API server
  cp             Copy files and directories to and from containers.
  auth           Inspect authorization
Advanced Commands:
  diff           Diff live version against would-be applied version
  apply          Apply a configuration to a resource by filename or stdin
  patch          Update field(s) of a resource using strategic merge patch
  replace        Replace a resource by filename or stdin
  wait           Experimental: Wait for a specific condition on one or many resources.
  convert        Convert config files between different API versions
  kustomize      Build a kustomization target from a directory or a remote url.
Settings Commands:
  label          Update the labels on a resource
  annotate       Update the annotations on a resource
  completion     Output shell completion code for the specified shell (bash or zsh)
Other Commands:
  api-resources  Print the supported API resources on the server
  api-versions   Print the supported API versions on the server, in the form of "group/version"
  config         Modify kubeconfig files
  plugin         Provides utilities for interacting with plugins.
  version        Print the client and server version information
Usage:
  kubectl [flags] [options]
```