Inhaltsverzeichnis

1 Einleitung

Anbieter von Cloud-Services haben den Anspruch, möglichst geringe und kurze Ausfallzeiten mit ihren Services zu erreichen. Um eine maximal lange, störungsfreie Servicelaufzeit zu erreichen, ist es notwendig jederzeit den Servicestatus einsehen und mögliches Fehlverhalten frühzeitig erkennen zu können. Da für eine dauerhafte Kontrolle eines Services ein oder sogar mehrere Mitarbeiter benötigt würden, welche eine eintönige Kontrollaufgabe übernehmen müssten, ist es sinnvoll möglichst viele Teile der Kontrolle zu automatisieren. Diese Automatisierung bringt einerseits den Vorteil der Kosteneinsparung, da keine Mitarbeiter für diese Aufgabe benötigt werden und andererseits einen Geschwindigkeitsvorteil durch die wesentlich geringere Reaktionszeit, die durch die Geschwindigkeit von Computern gegenüber dem Menschen einhergeht.
Hierbei sollen vor allem Engpässe bei Ressourcen ausfindig gemacht werden, sowie Anomalien, also Fehlverhalten, in einzelnen Komponenten der Infrastruktur gefunden und behoben werden, bestenfalls noch bevor sich größere Auswirkungen auf die restlichen Komponenten ergeben. Sofern Ressourcenengpässe, also hohe Last, auftritt und dies frühzeitig erkannt wird, können einzelne Services gezielt skaliert und so Beeinträchtigungen auf die Funktion verhindert werden. Da Engpässe oft temporär auftreten, werden Services sowohl hoch- als herunterskaliert um so immer die ideale Zahl an Ressourcen bereitgestellt. Des Weiteren soll Fehlverhalten detektiert werden. Dies liegt dann vor, wenn hohe Ressourcenlast ohne erkennbaren Grund vorliegt. Das wäre beispielsweise dann der Fall, wenn die Prozessorlast oder Speicherlast eines Services auf einem sehr hohen Wert läuft, gleichzeitig aber keine hohe Netzwerklast durch Nutzer vorliegt, die dieses Verhalten begründet. In diesem Fall kann von einer anomalen Funktion ausgegangen und ein Service neu, im besten Fall unterbrechungsfrei, bereitgestellt werden.
Um eine automatisierte Erkennung zu ermöglichen, werden Daten sog. Metriken benötigt, die eine Entscheidung auf Basis des vorliegenden Verhaltens treffen lassen. Metriken müssen erhoben und ausgewertet werden, um eine Aktion aus ihnen schließen zu können, welche die vorliegende Anomalie oder den vorliegenden Engpass beheben kann.
Die erhobenen Metriken müssen einerseits für Menschen lesbar sein, um aktuelle Zustände widerspiegeln und entsprechend darauf reagieren zu können, andererseits ebenso für Computer auswertbar sein, um die Automatisierung durch diese zu ermöglichen.
Neben der Möglichkeit automatisierte Aktionen auszuführen, ist es auch sinnvoll entsprechende verantwortliche Administratoren über das Fehlverhalten in Kenntnis zu setzen und diese zu benachrichtigen, um ihnen die Möglichkeit zu geben dem Verhalten auf den Grund zu gehen.
Diese Arbeit setzt sich das Ziel die Durchführbarkeit der automatisierten Anomalie- und Engpasserkennung nachzuweisen und die erste Implementierung innerhalb eines schon bestehenden Kubernetes-Clusters. Im Rahmen dieser Arbeit werden die passenden Komponenten gewählt, die zur Umsetzung der Anforderungen benötigt werden, die Infrastruktur geplant, erstellt und die korrekte Funktion evaluiert.
Des Weiteren wird die Relevanz verschiedener erhobener Metriken in Bezug auf ihre Verwendbarkeit beim automatisierten Detektieren von Anomalien und Engpässen dargestellt und geklärt.

Es werden die weit verbreiteten Tools Prometheus und Grafana verwendet und durch Eigenentwicklungen ergänzt und so eine Infrastruktur geschaffen, welche die Anforderungen erfüllen kann.

1.1 Struktur

Diese Arbeit ist in neun Kapitel strukturiert. Nach diesem Kapitel, dem ersten, das die Arbeit einleitet, werden im nächsten Kapitel die Grundlagen und für das Verständnis der weiteren Arbeit benötigtes Wissen vermittelt. Das Kapitel vermittelt grundlegendes Wissen zu den verwendeten Tools. Im dritten Kapitel wird der Status Quo der Technik ermittelt und somit die technische Ausgangslage der Arbeit geklärt. Mit dem vierten Kapitel beginnt der praktische Teil des Projekts. Am Anfang dessen steht die Vorgehensweise beim Aggregieren von Daten aus einem Kubernetes-Cluster. Nachdem Daten aggregiert wurden, müssen diese verarbeitet und ausgewertet werden. Das Vorgehen hierbei wird in Kapitel fünf erläutert. Die ausgewerteten Daten haben den Zweck automatisierte Aktionen auszulösen. Wie passende Aktionen gewählt und angewandt werden, damit befasst sich das sechste Kapitel. Hier werden verwendete Komponenten, Architektur, Aktionen, Regeln sowie die passende Programmiersprache erläutert. Nachdem die geplanten Features implementiert sind, müssen diese auf ihre korrekte Funktion getestet werden. Im siebten Kapitel wird geklärt, wie die korrekte Funktion evaluiert wird, wie der Messaufbau gestaltet wurde und welche Komponenten auf welche Art und Weise getestet wurden.
Nach der Evaluation werden die Ergebnisse der Arbeit diskutiert. Wurden alle Ziele erreicht ? Warum wurde welche Entscheidungen getroffen ? Diese und weitere Fragen werden in der Diskussion in Kapitel acht diskutiert.
Zum Schluss, in Kapitel neun, wird ein Fazit aus dem zurückliegenden Projekt gezogen und ein möglicher Ausblick in die Zukunft gestellt.

2 Grundlagen

2.1 Kubernetes

Kubernetes, kurz "k8s", ist eine ursprünglich von Google entwickelte, mittlerweile aber quelloffene Software zur Orchestrierung und Deployment von containerisierten Anwendungen. Seit seiner Einführung 2014 hat Kubernetes ein starkes Wachstum erlebt und ist zum Quasistandard bei der Entwicklung von Cloud-Native Applikationen geworden. Diese Übersicht der Grundlagen spricht die für dieses Projekt benötigten Komponenten an, für weitergehende Informationen gibt es die Dokumentation[Referenz auf Kube-Doku]

Die Relevanz von Kubernetes lässt sich an der halbjährlichen Befragung der Community der Cloud-Native Computing Foundation ablesen: In der Umfrage stellte sich heraus, dass über 78 Prozent der 1337 Befragten Kubernetes verwendet.[Umfrage CNCF 2020 ergänzen]

Kubernetes ist eine mittlerweile bewährte Infrastruktur und bietet Software die nötig ist um zuverlässige und skalierbare verteile Systeme zu entwickeln. [1]

2.1.1 Master Worker Prinzip

Der Zweck eines Kubernetes-Clusters besteht darin viele, einzelne Computer als eine einzige Einheit zusammenarbeiten zu lassen. Ein Cluster besteht aus zwei verschiedenen Arten von Komponenten, die zusammenarbeiten: dem Node und dem Master. Der Master stellt den Verwalter im Cluster dar. Er koordiniert alle Vorgänge, trifft also Entscheidungen die von globaler Bedeutung für das gesamte Cluster sind. Beispielsweise startet er Komponenten oder schaltet sie ab, ist aber auch für Tasks wie die Zeitplanung zuständig. Er wird über die Kubernetes-API angesprochen und steht in direkter Verbindung mit den Nodes.[2] [3] Nodes wiederum stellen die Arbeiter dar, weshalb sie auch "worker"genannt werden. Deren Aufgaben bestehen darin Pods[siehe Relevante Komponenten] aufrecht zu erhalten und die Laufzeitumgebung bereitzustellen. Nodes halten auch die Container-Runtime bereit, welche dafür zuständig ist Containerisierte Anwendungen auszuführen und so das verteilen einer Anwendung auf beliebige Hardware möglich macht.

<evtl. hier nochmal Grafik Master/Node>

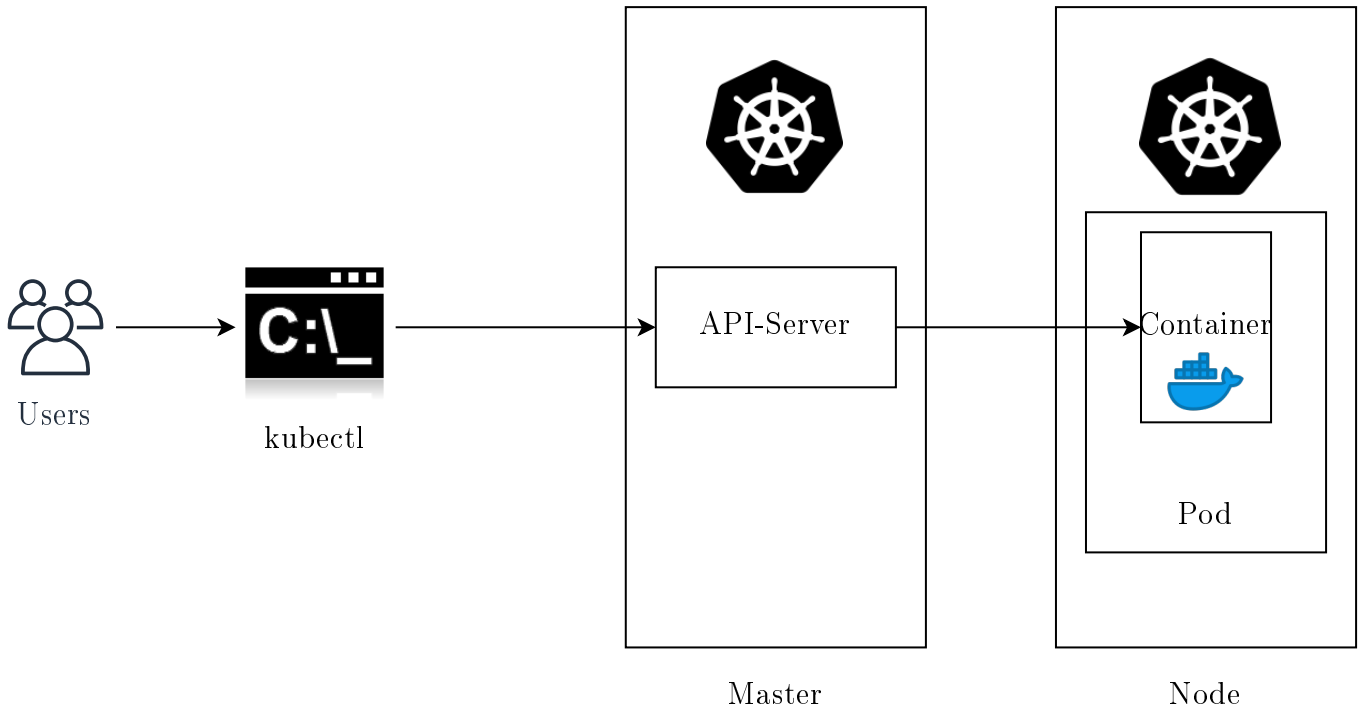

Abbildung 1: einfache Darstellung eines Kubernetes-Cluster (eigene Zeichnung)

2.1.2 Relevante Komponenten

Kubernetes bietet eine Vielzahl von Komponenten für unterschiedliche Aufgaben. Die für dieses Projekt Relevanten werden hier erklärt:

kubelet :
Der kubelet ist der primäre 'node-agent'. Er ist dafür zuständig die Nodes beim Kubernetes API-Server zu registrieren. Des Weiteren verwaltet er Pods anhand einer Podspezifikation(PodSpec) und sorgt dafür, dass die Pods im Rahmen der Spezifikation "gesund"laufen.
Über den kubelet können diverse Metriken gesammelt werden, die über den Status des Nodes oder der darin laufenden Pods und Container Auskunft geben.

Kubernetes bietet verschiedene Organisiationskonzepte mit deren Hilfe sich die Kubernetes-Struktur umsetzen lässt, die für dieses Projekt wichtigsten werden im Folgenden erläutert:

Pod :
Ein Pod ist eine 'execution unit' und repräsentiert einen Prozess, der in einem Cluster läuft. Ein Pod kapselt einen oder mehrere Anwendungs-Container, ein eigener Speicherbereich, eine eigene IP-Adresse sowie dessen Konfigurationsoptionen. Die meistverwendete Container-Runtime ist,wie auch in diesem Projekt, Docker, es

gibt aber auch Unterstützung für Weitere wie beispielsweise Rocket.

Deployment :
Das Deployment ist eine Beschreibung des Zustands, in dem Pods ausgeliefert werden sollen. Hier werden beispielsweise der Name, Namensraum, Ressourcenlimits oder auch die Größe der Skalierung definiert.

Service :
Nach der offiziellen Kubernetes-Dokumentation sind Services Eine äbstrakte Möglichkeit, eine Anwendung, die auf einer Reihe von Pods läuft, als Netzwerkdienst bereitzustellen". Kubernetes Pods können dynamisch in ihrer Anzahl skalieren und so auch ihre IP-Adresse wechseln. Daher ist es sinnvoll die Pods über einen Service anzusprechen, der mit einem DNS ähnlichen System funktioniert und so ein Deployment über einen lesbaren Namen ansprechbar macht.

kubectl :
kubectl ist eine Kontrollanwendung für Kubernetes. Es ist eine direkte Schnittstelle zwischen User und dem Kubernetes API-Server und kann mittels Konsolenbefehlen bedient werden. Das Tool stellt eine einfache Möglichkeit zur Bedienung des Kubernetes-Clusters dar.

Kubernetes bietet systemseitig Funktionen, die beim ermitteln von Cluster- und Systeminformationen helfen, eine davon ist der cAdvisor

cAdvisor :
Da Container von sich aus keine Informationen zu ihrem Ressourcenstatus nach außen preisgeben oder exportieren, bedarf es eines Hilfsmittels, das genau dies macht. cAdvisor(Container Advisor) ist ein Daemon, der Ressourcen-Informationen aus Containern sammelt, verarbeitet und exportiert.[4]

2.2 Metriken

Eine Metrik ist eine Funktion, die einen Zustand oder eine Eigenschaft als Maßzahl abbildet. Metriken in der Informatik lassen sich im Grunde in 3 Bereiche einteilen:

- Service-Metriken, welche die Performance eines Service bemessen, zum Beispiel die Unterbrechungsfreie Laufzeit.
- Prozess-Metriken, die für die Quantifizierung des Entwicklungsprozesses einer Software verwendet werden
- Technologie-Metriken, welche die zugrunde liegende Technologie quantifizieren, zum Beispiel die Speicherauslastung

Wenn Metriken über einen Zeitraum beobachtet werden und nach Messzeit strukturiert werden, werden sie Zeitreihen-Metriken genannt.
In dieser Arbeit werden Service-Metriken erstellt und verwendet, welche die Performance der in einem Kubernetes-Cluster laufenden Services beziffert.
<hier Quelle finden !>

2.3 Anomalie

2.4 Prometheus

Prometheus ist ein Open-Source Monitoring-Toolkit. Es wurde ursprünglich von SoundCloud entwickelt, ist aber mittlerweile ein Open-Source Projekt, das der Cloud Native Computing Foundation (CNCF) beigetreten ist. Die primären Funktionen des Toolkits sind das Aufzeichnen von Zeitreihen-Metriken und das Alarmieren bei Überschreitungen von Grenzwerten der Metriken. Des weiteren bietet Prometheus:

- Ein WebUI zum Visualisieren der aufgezeichneten Daten
- Eine eigene Abfragesprache(PromQL) für aufgezeichneten Metriken Regeln, Visualisierungen oder ähnliches erstellen zu können
- Einen Alertmanager um Alerts entgegen zu nehmen und weiter verwalten zu können
- Eine 'Target-Discovery' um selbstständig sinnvolle und parametrisierbare Ziele zu entdecken

Prometheus bietet eine Vielzahl an Komponenten, die das Überwachen von Systemen und Alarmieren unterstützen. Die grundlegende Komponente ist hierbei der zentrale Prometheus Server. Dieser ist dafür zuständig Metriken zu sammeln, im Folgenden scrapen genannt, und zentral zu speichern, sofern es gewünscht ist.

2.4.1 Scraping

Der Begriff Scraping bezeichnet das Sammeln von Metriken durch den Prometheus-Server. In Prometheus funktioniert dies folgendermaßen: Ein Scraping-Target, beispielsweise ein Kubernetes-Node, besitzt einen sog. Exporter, der Metriken aus dem System ausließt und diese an einem HTTP-Endpunkt '<IP-Adresse/DNS-Name>/metrics' bereitstellt. Der Prometheus-Server findet entweder per automatischem Target-Disovery Mechanismus das Target oder wird per Konfiguration darauf eingestellt. Der Exporter auf dem Target hat im Normalfall einen Aktualisierungszeitraum ebenso wie der Prometheus-Server, sodass die Metriken automatisch aktualisiert werden und eine Zeitreihenmetrik erzeugt wird.

2.4.2 Abfragen

Das Abfragen von von Metriken wird mittels der prometheuseigenen, an SQL angelehnten Query-Language PromQL durchgeführt. PromQL-Requests werden an den Prometheus-Server gestellt, der die Requests prüft, verarbeitet und entsprechende Werte als Antwort zurückgibt. Die Abfragen können sehr einfach sein, indem beispielsweise nur der Name einer Metrik angegeben wird und so der entsprechende Wert zurückgeliefert wird. Requests können aber auch, ähnlich SQL, miteinander kombiniert werden um voneinander abhängige Werte abzufragen oder Werte über unterschiedliche Zeiträume zu erhalten.

Ein beispielhafter Request: Dieser Request der den Mittelwert der HTTP-Codes 401 des Kong API-Gateways über die letzten 5 Minuten bildet:

```
rate(http_status{code='401'}[5m])
```

In diesem Projekt werden PromQL-Abfragen vor allem für zwei verschiedene Zwecke verwendet:

- zum erstellen von Regeln, bei deren Erfüllung der Prometheus-Server einen Alert verschickt (siehe nächstes Kapitel Alerting)
- zum Visualisieren der Metriken in der Prometheus Web-UI oder in Grafana-Boards

2.4.3 Alerting

Das Alerting in Prometheus funktioniert mittels festgelegten Regeln. Diese werden in der PromQL-Sprache auf dem Prometheus-Server definiert und gespeichert. Sobald eine Regel erfüllt ist sendet der Server einen Alert an den Alertmanager, der diesen dann weiter verarbeiten kann.
Eine Alert-Regel besteht im Grunde aus zwei Teilen. Der erste Teil ist die PromQL-Bedingung, die erfüllt sein muss. Sobald diese erfüllt ist erhält der Alert den Status 'pending'. In diesem Status wird der Alert noch nicht versendet, sondern wartet auf das Erfüllen eines vorgegebenen Zeitwertes. Erst nach erfüllen des Wertes wird der Alert an den Manager versendet.

Der Alertmanager hat mehrere Möglichkeiten mit dem Alert umzugehen. Eine der Möglichkeiten ist das Weiterleiten an definierbare Ziele, beispielsweise an bestimmte E-Mail Adressen, Chat-Programme wie Slack oder auch an Webhooks bzw. HTTP-Endpunkte.

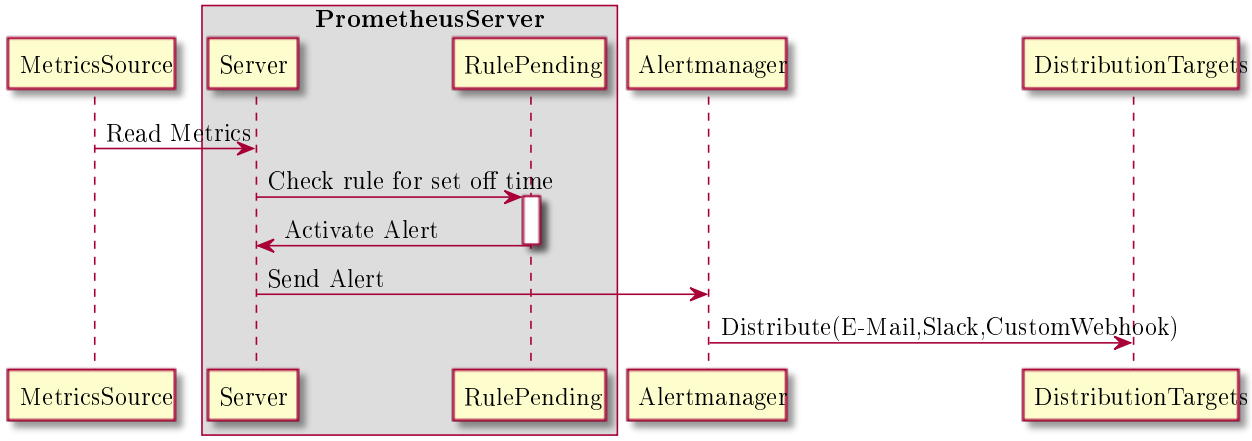

Abbildung 2: Alertmanager Aufbau (eigene Zeichnung)

3 Stand der Technik

3.1 Prometheus Fault-Detection-Centric Model

Das Buch "Monitoring with Prometheus"beschreibt auf Seite 18-19, dass Prometheus das "fault-detection-centric"model ganz Oben auf die Liste schreibt. Echtzeitmetriken geben eine guten Aufschluss darüber, welchen Zustand das Cluster im Moment hat.

3.2 Aktionen

3.2.1 Anomaly Detection

In ihrem Paper 'Anomaly Detection and Diagnosis for Container-based Microservices with Performance Monitoring' beschreiben Qingfeng Du et. al eine Möglichkeit mithilfe der Performance und Hardware-Metriken von Container basierten Microservices und Machine Learning Techniken Anomalien zu erkennen und so Service Level Agreement valuations(SLAV) zu reduzieren bzw. zu verhindern. Das Anomaly Detection System, folgend ADS abgekürzt, besteht aus drei verschiedenen Modulen. Diese Module sind das Monitoring-Modul, das dafür zuständig ist Performance Daten aus dem Zielsystem auszulesen, das Data-Processing Modul das die ausgelesenen Daten auf Anomalien zu prüfen sowie zuletzt das 'Fault-Injection' Modul, das Fehlerfälle im System erzeugt und so einen Datensatz erhält, der dafür genutzt wird das Machine-Learning Model anzulernen und zu validieren.
Die verschiedenen beschriebenen Arten der Anomalien, die das System abdeckt sind 'CPU-Hog's, Memory Leaks oder der Package-Loss von Containern. Die beiden Hauptaufgaben des Systems sind daher, die Klassifizierung, ob in einem Microservice eine Anomalie vorliegt und falls dies der Fall ist zu lokalisieren, wo diese stattfindet.
Dieses Paper bezieht sich ausschließlich auf die Detektion der Anomaly, nicht aber auf das Beheben dieser.

Weitere Paper verwenden ebenfalls Machine Learning Modelle wie beispielsweise 'A Controller Architecture for Anomaly Detection, Root Cause Analysis and Self-Adaptation for Cluster Architectures' von Areeg Samir et al. ein Hierarchical Hidden Markov Models oder 'KubAnomaly: Anomaly detection for the Docker orchestration platform with neural network approaches' Chin-Wei Tien et al., das in dem Journal 'Engineering Reports' veröffentlicht wurde.

Im Rahmen dieses Projekts wird gegenüber den genannten Papers ein anderer, mittlerweile vor allem in Papers eher seltener zu findender, aber aus diversen Gründen interessanter Ansatz gewählt. Der Ansatz, der in diesem Projekt gewählt wird, basiert auf Regeln mit statischen Thresholds. Dieser verspricht gegenüber den Machine-Learning Ansätzen einige Vorteile die es zu untersuchen vor Allem die leichte Erweiterbarkeit des System durch einfaches Erweitern von Regeln und die Einsatzmöglichkeit für Firmen, die keine Expertise in Machine Learning besitzen.

3.2.2 Cluster-Skalierung

In dem Paper 'ACCRS: autonomic based cloud computing resource scaling' geschrieben von Ziad A. Al-Sharif et al. wird ein Skalierungssystem für Cloud-Ressourcen beschrieben. Dieses arbeitet mithilfe statischer Regeln und Thresholds.
Das System besteht aus mehreren Komponenten, die Erste, welche die Basis des Ganzen bildet ist die 'System State Monitoring' (SSM) Komponente. Sie zeichnet CPU, RAM, Netzwerk Utilization, und vieles mehr auf. Um aus den aufgezeichnete Monitoringdaten Entscheidungen und entsprechende Funktionen auszuführen gibt es das Modul 'System state analyses and decision making algorithm'(SSA-DMA). Das SSA-DMA Modul bietet die zwei Algorithmen, der Erste stellt den Systemdurchsatz der Anzahl der verwendeten VMs gegenüber um so Probleme mit der Hardware zu finden. Im Falle eines erkannten Problems wird ein Root-Cause-Analysis Algorithmus ausgeführt um den fehlerhaften Host auszutauschen.
Der zweite Algorithmus im SSA-DMA Modul ist der 'Workload Classification Algorithm (WCA)'. Dieser sorgt dafür, dass ein System immer die optimale Anzahl an VMs und Ressourcen zur Verfügung hat.
Der Algorithmus funktioniert so, dass er die Auslastung eines Systems als hoch oder niedrig einstuft. Mittels Messen der Utilization von CPU,RAM und Netzwerk und Abgleichen mit den Thresholds wird identifiziert, ob die Systemressourcen skaliert werden müssen. Das Skalieren hat den Zweck ein System durch aufwenden zusätzlicher Ressourcen die Auslastung in die sogenannte 'Safe-Zone' zu bringen. Die Safe-Zone beschreibt den Bereich zwischen 70%-80% Utilization der Ressourcen. Dieser Bereich wird als Bereich der idealen Auslastung beschrieben, da hier weder die Ressourcenlast zu Nahe am Leistungslimit liegt, noch so niedrig ist, dass zu viele Ressourcen verwendet werden, die im Zweifelsfall vermeidbare Kostenaufwände bedeuten können.
In dem Paper werden so in Summe 5 Zustände beschrieben in denen sich das System durch die beiden Algorithmen befinden kann, diese sind:

Safe zone Die Safe-Zone ist der Idealzustand, in dem das System sich befinden kann. Dieser Zustand ist erreicht, wenn sich die Utilization zwischen 70%-80% befindet.

Hier ist der Utilization Level, der Energieverbrauch und die 'Quality of Service' im Optimum. Alle weiteren Zustände zielen darauf ab, den Zustand der Safe-Zone herzustellen.

Under-utilization (UU) Der Zustand 'Under-utilization' tritt ein, wenn das System eine geringe Auslastung seiner Ressourcen feststellt. Der Energieverbrauch ist in diesem Zustand hoch, während der Durchsatz gering ist.

Under-utilization with fault (UUF) Dieser Zustand tritt ein, wenn die Bedingungen eine UU-Zustandes erfüllt sind, außerdem aber auch noch ein fehlerhafter Zustand vorliegt(bspw. defekte Hardware)

Over-utilization (OU) Der Zustand der 'Over-utilization' tritt ein, wenn das System eine Überlastung seiner Ressourcen feststellt. Dieser Zustand wird bei einer Ressourcenauslastung über 80% erreicht und kann dafür sorgen, dass eingehende Workloads verzögert oder sogar verworfen werden.

Over-utilization with Fault (OUF) Dieser Zustand wird erreicht, wenn die Voraussetzungen des OU-Zustands erfüllt sind, außerdem aber auch noch ein fehlerhafter Zustand vorliegt(bspw. defekte Hardware)

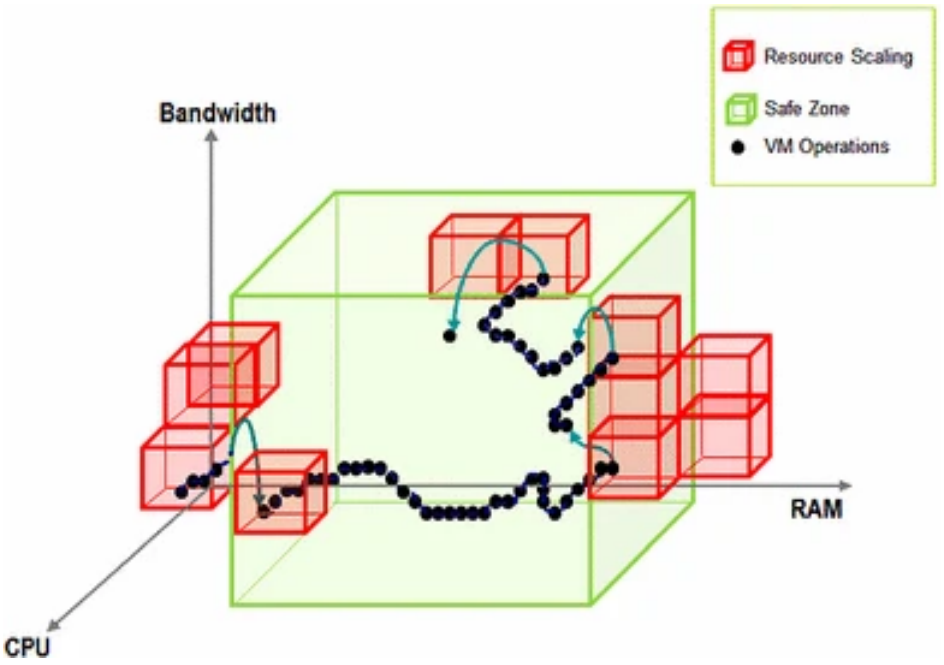

Abbildung 3: Safe-Zone and Resource Scaling Thresholds (from Paper ACCRS: autonomic based cloud computing resource scaling)

Die Skalierung in diesem Projekt orientiert sich an dem Verfahren, welches im Paper 'ACCRS: autonomic based cloud computing resource scaling' eingesetzt wird. Die

Idee der Zustände Safe-Zone, Under-Utilization sowie Over-Utilization und deren Erkennung mittels statischer Regeln werden adaptiert und auf das Kubernetes-Konzept der Deployment-Skalierung übertragen, die Thresholds allerdings abgewandelt.

Literatur

[1] Brendan Burns, Joe Beda, and Kelsey Hightower. *Kubernetes: Up and running : dive into the future of infrastructure*. O'Reilly Media, Incorporated, Sebastopol, CA, second edition edition, 2019.

[2] Kubernetes komponenten, 30.05.2020.

[3] Using minikube to create a cluster, 16.03.2020.

[4] Github-User: dashpole. google/cadvisor, 05.07.2020.