Fachhochschule Nordwestschweiz
Hochschule für Technik

Energyefficient Kubernetes

Bachelor Thesis

Windisch, 22. März 2024

Student/Studentin	Jérôme Baur Julie Engel
Fachbetreuer	Prof. Dr. Sebastian Graf
Fachexperte	Christoph Eberle, Red Hat
Auftraggeber	Norwin Schnyder, SIX Group
Projektnummer	23HS_IMVS2

Fachhochschule Nordwestschweiz, Hochschule für Technik

Abstract

Cloud Computing hat sich als unverzichtbarer Bestandteil der IT-Welt etabliert. Es ermöglicht Unternehmen, Ressourcen wie CPU und Arbeitsspeicher flexibel und bedarfsgerecht zu nutzen. Eine effiziente Nutzung dieser Ressourcen ist entscheidend, da eine ineffiziente Nutzung mit einem erhöhten Stromverbrauch einhergeht. Bei der Produktion von Strom werden, direkt oder indirekt, Treibhausgase emittiert. Aufgrund des kontinuierlichen und starken Wachstums von Kubernetes als Schlüsseltechnologie besteht ein erhebliches Potenzial, die CO_2-Emissionen durch eine effizientere Nutzung dieser Plattform zu reduzieren.

Diese Arbeit untersucht den Energieverbrauch von Kubernetes-Clustern mithilfe des Kepler-Projektes und zielt darauf ab, Strategien zur Steigerung der Energieeffizienz zu identifizieren und zu bewerten. Im Zentrum stehen dabei zwei Hauptansätze: der Vergleich von Monolithen und Microservices hinsichtlich ihres Energieverbrauchs und die Evaluation des Einsatzes des Vertical Pod Autoscalers (VPA) zur Optimierung der Ressourcennutzung.

In der Untersuchung des ersten Ansatzes wurde festgestellt, dass die Deploymentstrategie (Monolith vs. Microservices) allein keine signifikante Auswirkung auf die Energieeffizienz hat. Der Energieverbrauch wird hauptsächlich durch den Idle Energieverbrauch der Nodes beeinflusst.

Der zweite Ansatz dieser Arbeit, der Einsatz des Vertical Pod Autoscalers, zeigte hingegen positive Effekte auf die Energieeffizienz. Durch die dynamische Anpassung der Ressourcenanforderungen von Pods und durch das Entfernen von Nodes konnte die Auslastung der Nodes verbessert und somit der Idle Energieverbrauch verringert werden. Dies führte zu einer signifikanten Reduktion des Energieverbrauchs.

Diese Arbeit bietet wichtige Einsichten in die Energieeffizienz von Kubernetes-Clustern und zeigt auf, dass insbesondere die Reduktion des Idle Energieverbrauchs und der gezielte Einsatz von Ressourcenmanagement-Tools entscheidend für effizientere Ressourcennutzung sind.

Keywords: Energyefficient Kubernetes, CO_2-Emissionen, Kepler.

Vorwort / Danksagung

Diese Arbeit wurde als Teil des Informatik-Studiums an der Fachhochschule Nordwestschweiz (FHNW) in Brugg entwickelt und von der SIX Group initiiert. Norwin Schnyder diente dabei als Ansprechpartner seitens der SIX Group. An dieser Stelle möchten wir uns bei ihm für die vielen Anregungen und die Unterstützung bedanken.

Ebenfalls bedanken möchten wir uns bei unserem Betreuer der FHNW, Prof. Dr. Sebastian Graf, der uns fortlaufend mit vielen Inputs und Tipps durch die Arbeit begleitet hat und uns jederzeit für unsere Fragen zur Verfügung stand.

Ein besonderer Dank geht an Rita Baur, welche das Korrekturlesen der Arbeit übernommen und bei Fragen zu Formulierungen stets geholfen hat.

Brugg, im März 2024

Julie Engel, Jérôme Baur

Inhaltsverzeichnis

1 Einleitung

In diesem Kapitel werden kurz die Problemstellung und die Ziele dieser Arbeit aufgezeigt. Weiter wird der Auftraggeber SIX vorgestellt und die Vorgehensweise dieser Arbeit erläutert.

1.1 Problemstellung

In der heutigen Zeit hat sich Cloud Computing als unverzichtbare Technologie etabliert, welche in der Informationstechnologie (IT) zunehmend an Bedeutung gewinnt[1]. Die Möglichkeit, Ressourcen wie CPU und Arbeitsspeicher bedarfsgerecht zu beziehen und dynamisch anzupassen, bringt zahlreiche Vorteile wie eine schnellere Skalierbarkeit und Bereitstellung oder bessere Kosteneffizienz mit sich. Diese Flexibilität birgt jedoch auch Herausforderungen im Hinblick auf die Effizienz der Ressourcennutzung. Eine ineffiziente Nutzung führt nämlich nicht nur zu einem unnötig hohen Energieverbrauch, sondern auch zu einer erhöhten Freisetzung von Treibhausgasen und beschleunigt den Klimawandel.

Angesichts der globalen Bestrebungen zur Reduzierung von CO_2-Emissionen gewinnt die Optimierung des Energieverbrauchs in der IT-Branche zunehmend an Bedeutung, insbesondere im Bereich des Cloud-Computings[2]. Unternehmen erkennen zunehmend die Notwendigkeit, ihre CO_2-Bilanz zu verbessern. Dies nicht nur aus Umweltschutzgründen, sondern auch als Massnahme zur Kostensenkung und zur Stärkung des Unternehmensimages.

Bereits heute machen Datenzentren alleine ungefähr 2–3% des globalen Stromhaushalts aus[3]. Es wird aber davon ausgegangen, dass dieser Verbrauch noch in diesem Jahrzehnt drastisch nach oben gehen wird[3].

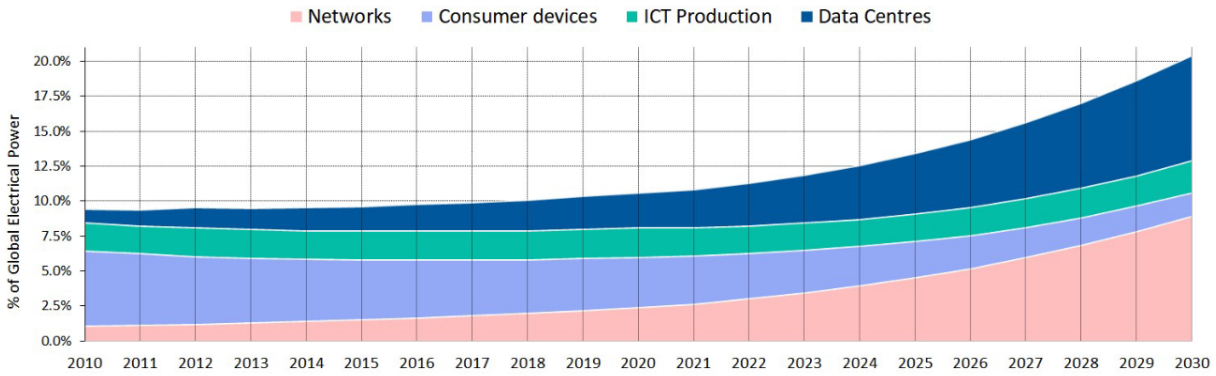


Abbildung 1.1: Erwarteter globaler Stromverbrauch von 2010–2030 [3]

Abbildung 1.1 zeigt den bisherigen und den zu erwartenden Energieverbrauch in Prozent des globalen Energieverbrauchs für Netzwerke, Benutzergeräte, ICT Produktion und Datenzentren. Die y-Achse ist der prozentuale Anteil des globalen Energieverbrauchs und die x-Achse der Zeitstrahl von 2010–2030.

Gemäss der Abbildung 1.1 wird davon ausgegangen, dass Datenzentren im Jahr 2030 ca. 8% des globalen Stromverbrauchs ausmachen werden[3]. Da dies ein nicht unbedeutender Anteil ist, ist es wichtig, dass Datenzentren in Zukunft energieeffizient operieren.

Aufgrund dieser Tatsachen fokussiert sich die vorliegende Bachelorarbeit auf die Steigerung der Energieeffizienz beim Einsatz von Kubernetes, einer führenden Technologie im Bereich des Cloud-Computings.

1.2 Ziele

Das primäre Ziel dieser Arbeit ist es, durch eine detaillierte Analyse und Versuche zur Optimierung des Ressourcenverbrauchs bei gleichbleibender Systemleistung herauszufinden, welche Ebenen und Faktoren beim Cloud-Computing den grössten Einfluss auf Energieersparnis haben.

Konkret sollen am Ende dieser Arbeit die folgenden Fragen beantwortet werden:

- Auf welcher Ebene (Deploymentstrategie / Architektur) hat man die grössten Einflussfaktoren in Bezug auf Energieersparnis bei gleichbleibender Performance?
- Welche Faktoren haben aus Sicht einer auf Kubernetes deployten Applikation den meisten Einfluss in Bezug auf die Energieersparnis?

Die Erkenntnisse dieser Arbeit sollen einerseits einen Beitrag zum Umweltschutz leisten, andererseits Unternehmen wie SIX bei der Erreichung ihrer Klimaziele unterstützen.

1.3 SIX Group

Der Auftraggeber dieser Arbeit, die SIX Group («SIX»), ist Finanzdienstleister und betreibt die Infrastruktur für die Finanzplätze in der Schweiz und in Spanien sowie deren Börse. Das Unternehmen erbringt Dienstleistungen rund um Wertschriften, Finanzinformationen und den Zahlungsverkehr[4].

Auch in der SIX hat in den letzten Jahren Kubernetes als Cloud Computing-Technologie stark an Bedeutung gewonnen. So sind momentan über 20 Cluster mit insgesamt mehr als 300 Nodes in Betrieb (Erklärung der Begriffe Cluster und Nodes folgt im Kapitel 2.1.1).

SIX hat sich anlässlich des Zero Emissions Day 2022 dazu verpflichtet, bis im Jahr 2050 das Ziel «Net Zero CO_2 Emissions» zu erreichen[5]. Seit 2016 veröffentlicht SIX jährlich ein Sustainability Report, welcher transparent den aktuellen Stand und die Bemühungen dazu rapportiert.

1.4 Methodik

In dieser Arbeit wird als Erstes zum besseren Verständnis eine Wissensgrundlage geschaffen, welche in Kapitel 2 dokumentiert wird. Ansätze von bisherigen Forschungsgruppen werden in Kapitel 3 vorgestellt und deren Relevanz in Kapitel 3.3 diskutiert.

Um die Ziele aus Kapitel 1.2 zu erreichen, werden verschiedene Ansätze als Experimente durchgeführt, die im Hauptteil dieser Arbeit, in Kapitel 4 dokumentiert sind: Zunächst wird in Kapitel 4.1 ein Tool installiert, um den Energieverbrauch eines Clusters möglichst genau ermitteln zu können, anschliessend werden in den Nachfolgenden Kapitel 4.2 und 4.3 zwei unterschiedliche Ansätze zur Optimierung der Energieeffizienz verfolgt.

Die daraus gezogenen Erkenntnisse und Zukunftsaussichten werden in Kapitel 5.1, respektive 5.2 erläutert und diskutiert.

2 Grundlagen

Dieses Projekt beruht auf der Basis von Kubernetes und Kepler, weshalb in diesem Kapitel die Grundlagen dieser beiden Technologien erläutert werden.

Im Kapitel 2.1 wird erläutert, was unter Kubernetes als Infrastruktur-Plattform verstanden wird und wie die unterschiedlichen Komponenten aufgebaut sind. Anschliessend wird im Kapitel 2.2 auf Kepler eingegangen und wie Kepler verwendet wird, um den Energieverbrauch zu messen.

2.1 Kubernetes

Kubernetes ist ein Open-Source-System zur Orchestrierung, Skalierung und Verwaltung von containerisierten Anwendungen[6]. Es abstrahiert die darunterliegenden Infrastrukturelemente wie Speicher oder Netzwerk und erlaubt, eine deklarative Konfiguration via YAML-Files sowie die Möglichkeit, Befehle imperativ auszuführen.

Kubernetes wurde ursprünglich von Google entwickelt und ist mittlerweile ein Projekt der Cloud Native Computing Foundation (CNCF).

2.1.1 Aufbau eines Clusters

Ein Cluster besteht aus einer Reihe von physikalischen oder virtuellen Servern, die als Nodes bezeichnet werden. Gemeinsam stellen sie die erforderlichen Ressourcen für das Ausführen containerisierter Anwendungen bereit.

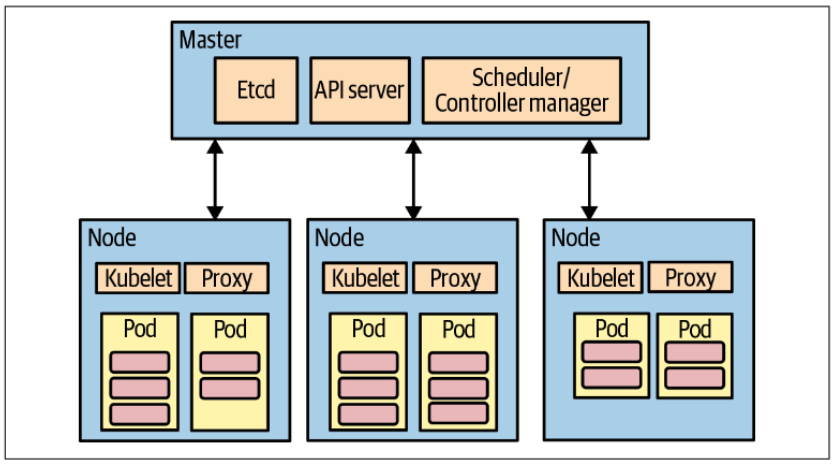


Abbildung 2.1: Komponenten eines Kubernetes-Clusters. Quelle: [7, S. 11]

Wie in Abbildung 2.1 ersichtlich, gibt es zwei Arten von Nodes: Master- und Worker-Nodes[8]. Sie haben unterschiedliche Arbeitsgebiete und bestehen aus unterschiedlichen Komponenten.

Master-Nodes, auch «Control Plane» genannt, koordinieren den Cluster, indem sie für das Scheduling, das Skalieren von Anwendungen und das Verwalten der Nodes verantwortlich sind.

Dazu werden verschiedene Komponenten auf den Master-Nodes eingesetzt:

- ein API-Server, welcher von allen Nodes angesprochen wird
- eine etcd-Instanz, welche alle Konfigurationen verwaltet
- ein kube-scheduler, welcher für das Scheduling der Workloads verantwortlich ist
- ein kube-controller-manager, welcher registrierte Controller administriert

Worker-Nodes sind für die Ausführung der Container-Anwendungen verantwortlich. Jeder Worker-Node ist mit einem `kubelet` ausgestattet, welches mit den Master-Nodes kommuniziert und sicherstellt, dass die Workloads (siehe Kapitel 2.1.2) wie vorgesehen ausgeführt werden.

Die Aufteilung in mehrere Master- und Worker-Nodes ermöglicht eine effiziente Verteilung von Aufgaben und Ressourcen. Sie trägt zu einer hohen Verfügbarkeit und einer effizienten Leistung des Clusters bei.

2.1.2 Workloads

In Kubernetes spielen Workloads eine zentrale Rolle. Sie definieren die eigentliche Ausführung von Anwendungen in Containern. Workloads werden durch verschiedene Ressourcentypen wie Deployments, DaemonSets und Pods organisiert[9].

Deployments sind eine Methode, um Zustände von Anwendungen zu deklarieren. Diese gewünschten Zustände erreichen sie, indem sie sicherstellen, dass Pods laufen und aktualisiert werden.

DaemonSets stellen sicher, dass auf jedem Node des Clusters eine Kopie eines Pods läuft. Dies ist für systemnahe Dienste wie Log-Collectors oder Monitoring-Agents ideal.

Pods sind die kleinste Einheit in Kubernetes. Sie kapseln einen oder mehrere Container, die eng zusammengehörende Anwendungen enthalten und die gemeinsam Ressourcen wie Netzwerk und Speicher teilen. Innerhalb eines Pods laufen Container, welche die eigentlichen Anwendungen ausführen. Siehe dazu auch Abbildung 2.1, in welcher Pods in Gelb markiert sind, und einzelne Container in Rot.

Um eine bessere Isolation und Verwaltung von Ressourcen bereitzustellen, bietet Kubernetes *Namespaces* an. Diese ermöglichen es, Ressourcen in logisch voneinander getrennte Gruppen zu organisieren. Namespaces sind besonders nützlich in Umgebungen mit vielen Teams oder Projekten, da sie eine bessere Kontrolle über Workloads und Zugriffsrechte erlauben.

2.1.3 Scheduling

Der `kube-scheduler` ist der Standard-Scheduler für Kubernetes[10]. Er ist so aufgebaut, dass falls notwendig oder gewünscht, der Scheduler erweitert oder sogar vollständig ersetzt werden kann. So können benutzerdefinierte Scheduling-Strategien gemäss spezifischen Anforderungen implementieren werden.

Die Mächtigkeit des Kubernetes Scheduler liegt in seiner Fähigkeit, komplexe Entscheidungen in Echtzeit zu treffen, um die Effizienz, Skalierbarkeit und Zuverlässigkeit von Anwendungen zu maximieren. Er bewertet die Anforderungen jedes Pods und die Verfügbarkeit der Nodes, um eine optimale Platzierung zu gewährleisten.

Beim Scheduling ordnet der Scheduler die Workloads jeweils den bestgeeigneten Nodes zu, damit das `kubelet` die Pods ausführen kann[10]. Dabei überwacht der Scheduler die neu erstellten Pods, die noch keinem Node zugewiesen sind.

Gemäss der Dokumentation[10] des `kube-scheduler` wird mittels zweier Methodiken versucht, den jeweils besten Node für einen Pod zu evaluieren:

- Beim *Filtering* schaut der Scheduler, ob ein Node die Anforderungen eines Pods erfüllt und filtert jene aus, welche diese Anforderungen nicht erfüllen. Entspricht kein Node den Anforderungen des Pods, so wird der Pod nicht gescheduled.
- Das *Scoring* bewertet danach, basierend auf den aktiven Scoring-Regeln, alle nicht ausgefilterten Nodes und wählt denjenigen mit dem höchsten Wert aus.

2.1.4 Ressourcenverwaltung

Das Konzept von Requests und Limits[11] ist fundamental für die Ressourcenverwaltung der Container, die innerhalb eines Clusters ausgeführt werden.

Bei der Definition eines Pods kann man mittels Requests und Limits optional angeben, wie viel Ressourcen ein Container benötigt. Die am häufigsten angegebenen Ressourcen sind CPU und Arbeitsspeicher (RAM).

Requests definieren die Mindestmenge an Ressourcen (CPU und Speicher), die einem Container garantiert zur Verfügung gestellt werden. Wenn ein Pod gestartet wird, reserviert das `kubelet` die angeforderten Ressourcen für diesen Pod auf dem Node, um sicherzustellen, dass diese Mindestanforderungen immer erfüllt sind. Die Scheduling-Entscheidung, auf welchem Node ein Pod platziert wird, berücksichtigt Requests, um Überbuchungen zu vermeiden und eine angemessene Ressourcenverfügbarkeit zu gewährleisten.

Limits legen die maximale Menge an Ressourcen fest, die ein Container verbrauchen darf. Wenn ein Container mehr Ressourcen als sein Limit verbraucht, kann das System eingreifen, zum Beispiel durch das Beenden eines Containers im Falle von Speicherüberschreitung oder durch Drosselung der CPU-Nutzung. Limits verhindern Ressourcenüberlastung auf Nodes.

Konzepte wie HPA und VPA ergänzen dabei diese Mechanismen, indem sie eine dynamische Skalierung und Ressourcenanpassung basierend auf aktuellen Lastbedingungen bieten.

Ein *Horizontal Pod Autoscaler* (HPA) skaliert automatisch die *Anzahl der Pods* in einem Workload basierend auf beobachteter CPU-Nutzung oder anderen ausgewählten Metriken.

Ein *Vertical Pod Autoscaler* (VPA) passt automatisch Requests und Limits für CPU und Speicher der Pods basierend auf deren Nutzung an, um eine optimale Ressourcennutzung zu gewährleisten.

2.1.5 Operators und Controllers

Operators[12] und Controllers[13] sind Schlüsselkonzepte in Kubernetes, welche die automatische Verwaltung und Steuerung eines Clusters ermöglichen.

Ein *Controller* ist eine Dauerschleife, welche einen gewünschten Zustand beobachtet und Massnahmen ergreift, um den aktuellen Zustand dem gewünschten Zustand anzugleichen. So kümmert er sich beispielsweise um Ressourcen wie DaemonSets (siehe Kapitel 2.1.2), in welchen er sicherstellt, dass auf jedem Node eine Kopie des gewünschten Pods läuft.

Ein *Operator* ist eine Erweiterung dieses Konzepts. Er nutzt die Kubernetes-API, um benutzerdefinierte Ressourcen zu erstellen und zu verwalten. Die Verwendung von Operators erlaubt es, das Wissen über das Laufzeitverhalten einer Applikation in Software zu kapseln. Damit ist es möglich, komplexere und zustandsbehaftete Anwendungen auf Kubernetes zu betreiben, als ob sie native Kubernetes-Komponenten wären.

2.1.6 Helm Charts

Ein Helm Chart ist im Grunde ein Paket aus vordefinierten Kubernetes-Ressourcen[14]. Sie ermöglichen die Installation und Konfiguration voneinander abhängiger Ressourcen und bieten damit eine einfache Möglichkeit, Anwendungen und ihre Abhängigkeiten als ein Ganzes zu verwalten. Durch den Einsatz von Variablen und Templating-Mechanismen unterstützen sie Wiederverwendung und reduzieren dabei durch manuelle Konfiguration hervorgerufene Fehler.

2.2 Kepler

Kepler (Kubernetes-basierter Efficient Power Level Exporter) ist ein Projekt, welches aus dem *Kepler Exporter* und dem *Kepler Model Server* besteht[15]. Es bietet die Möglichkeit, den Stromverbrauch eines Kubernetes-Clusters auf Prozess-, Container- und Pod-Ebene je nach Installationsumgebung entweder zu messen oder zumindest abzuschätzen.

Der *Kepler Exporter* ist ein Prometheus Exporter (siehe Kapitel 2.2.2), welcher Technologien wie Extended Berkeley Packet Filter (eBPF), Hardware-Counters und Linux-Kernel-Tracepoints nutzt (siehe Kapitel 2.2.1), um Daten von verschiedenen Systemkomponenten zu sammeln[16].

Diese Daten werden mithilfe von Machine Learning via dem *Kepler Model Server* verarbeitet, um den Energieverbrauch von Kubernetes-Pods abzuschätzen[16].

In dieser Arbeit wird der Kepler-Exporter verwendet, um Daten zum Energieverbrauch eines Kubernetes-Clusters zu bekommen[17]. Er kann als Operator oder via Helm Chart auf dem Cluster installiert werden[18]. Im Kapitel 4.1 wird auf die konkrete Installation eingegangen.

2.2.1 Vorgehensweise von Kepler

Gemäss der offiziellen Dokumentation[16] variiert die Erfassung des Energieverbrauchs von Systemen in Kubernetes-Umgebungen mittels des Kepler-Projekts je nach Einsatzumgebung in physischen Servern (Bare Metals) oder in virtuellen Maschinen (VMs).

In *Bare-Metal-Umgebungen* ermöglicht Kepler die direkte Erfassung von Echtzeit-Daten zum Systemenergieverbrauch mittels Hardware-Counters, basierend auf der erweiterten Konfigurations- und Stromversorgungsschnittstelle (ACPI) und dem Redfish/Intelligent Power Management Interface (IPMI). Kepler nutzt das Verhältnisleistungsmodell (Ratio Power Model), um den Energieverbrauch spezifischer Systemressourcen aufzuteilen.

In *VM-Umgebungen* kann der Energieverbrauch einer VM nicht direkt gemessen werden, sondern muss mittels dem Kepler Model Server geschätzt werden. Diese Modelle nutzen eBPF anstelle von Hardware-Countern, um das Verhältnis der Nutzung zu berechnen.

Das Verhältnisleistungsmodell (Ratio Power Model) teilt den Energieverbrauch über alle Prozesse, basierend auf deren Nutzung der Ressourcen im Verhältnis zur Gesamtnutzung des Systems.

Der Energieverbrauch jedes Prozesses wird berechnet, indem der Verbrauch durch das Verhältnis der Nutzung von Prozess und System geteilt wird. Die so ermittelten Daten für den Energieverbrauch von Prozessen werden dann auf Container- und Kubernetes-Pod-Ebene aggregiert. Es erfolgt eine Zuweisung, zu welchem Container ein Prozess gehört. Dies geschieht, indem die Prozess-ID mit der Container-ID korreliert wird, um diese den Namen der Pods zuzuordnen

2.2.2 Prometheus Exporter und Grafana

Ein Prometheus Exporter sammelt Metriken aus verschiedenen Systemen und Anwendungen und wandelt diese in ein Format um, das von Prometheus gelesen werden kann[19].

Prometheus ist ein Überwachungs- und Alarmierungstool, das speziell für dynamische Container-Umgebungen wie Kubernetes konzipiert wurde[20]. Exporter ermöglichen es Prometheus, Daten zu erfassen, die nicht nativ von Prometheus unterstützt werden.

Die durch Kepler via dem Monitoring-Tool Prometheus gesammelten Metriken zum Energieverbrauch können dann in einem Visualisierungstool wie Grafana dargestellt werden.

3 Stand der Forschung

Zum Thema energieeffizientes Kubernetes gibt es bereits eine Vielzahl von wissenschaftlichen Arbeiten, wovon sich viele in der Thematik überschneiden. Im folgenden Kapitel wird ein holistischer Scheduler vom Paper «Improving Data Center Efficiency Through Holistic Scheduling In Kubernetes» vorgestellt und im darauffolgenden Kapitel ein kohlenstoffarmer Scheduler vom Paper «A Low Carbon Kubernetes Scheduler».

3.1 Improving Data Center Efficiency Through Holistic Scheduling In Kubernetes

Die wissenschaftliche Arbeit von Townend et al.[3] verfolgt den Ansatz, den Standard-Scheduler mit einem eigenen holistischen Scheduler zu überschreiben.

Der Standard-Scheduler verteilt die Last gleichmässig auf alle Nodes, die darunter liegende Hardware kann jedoch unterschiedlich auf diese Last reagieren. Zum Beispiel können unterschiedliche Stufen von Hardware-Last je nach Hardware unterschiedliche Mengen an Hitze generieren. Deshalb muss die Hardware mithilfe von Energie unterschiedlich stark gekühlt werden.

Der holistische Scheduler versucht nun das System als Ganzes zu betrachten, daher der Name holistischer Scheduler. Je nach Anforderung minimiert er den Energieverbrauch oder erhöht die Performance.

Der Scheduler erreicht dies, indem er mit Vorhersagemodellen zum Verhalten der Hardware arbeitet. Basierend auf diesen Vorhersagen passt er das Scheduling an, um Energie zu sparen. Zudem sammelt das darunterliegende System Live-Metriken, die wiederum ins Vorhersagemodell gespiesen werden.

3.1.1 Resultate

Um die Effektivität des holistischen Schedulers zu messen, wurde das System in einem Datenzentrum in Schweden implementiert und getestet. In zwei durchgeführten Experimenten des Papers von Townend et al.[3] wurden zwei Modi des Default- und des holistischen Schedulers verwendet, nämlich ein *Performance* Modus und ein *Power-Saving* Modus. Beim Performance Modus war die Windkühlung immer zu 100% aktiv, während beim Power-Saving Modus die Kühlung dynamisch an die Auslastung angepasst wurde. In folgenden Experimenten bezeichnet Edgetic den holistischen Scheduler.

Im ersten Experiment wurden deterministische Workloads verwendet, um dem Modell die Vorhersage zu erleichtern. In diesem wurde der holistische Scheduler auf einem Cluster von 56 Servern angewendet. Es wurden drei Gegenüberstellungen gemacht.

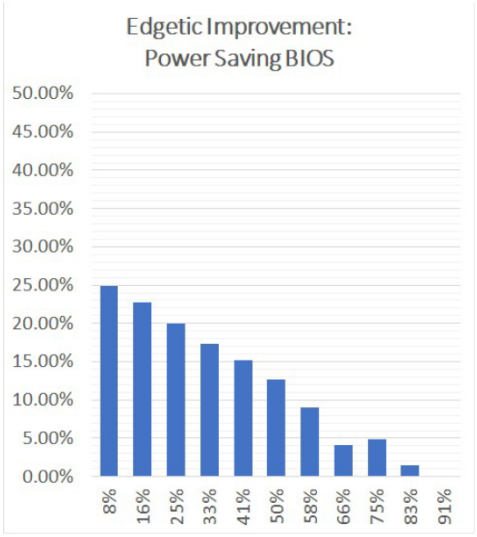


Abbildung 3.1: Energieersparnis Power-Saving Modus: Edgetic im Vergleich mit dem Default-Scheduler [3]

In der Abbildung 3.1 wird der Edgetic-Scheduler dem Default-Scheduler, beide im Power-Saving Modus, gegenübergestellt. Die x-Achse stellt die prozentuale Prozessorauslastung dar. Auf der y-Achse ist die prozentuale Energieersparnis des holistischen Schedulers verglichen mit dem Standard-Scheduler abgebildet. Die Abbildung 3.1 zeigt einen interessanten Trend in diesem Experiment auf: Je geringer die prozentuale Prozessorauslastung, desto höher ist die Energieersparnis. Hingegen bei einer Prozessorauslastung von 91% gibt es sogar keine Energieersparnis.

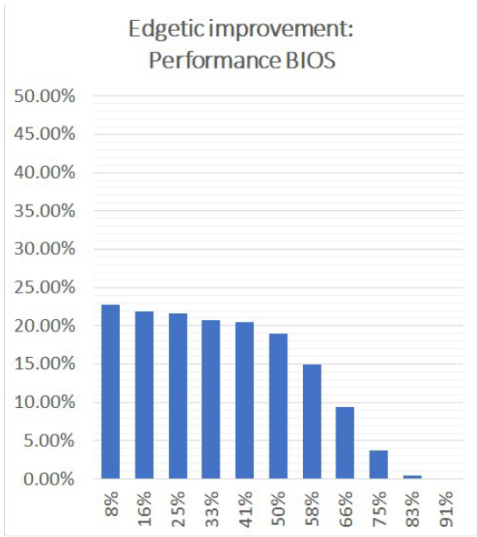


Abbildung 3.2: Energieersparnis Performance Modus: Edgetic im Vergleich mit dem Default-Scheduler [3]

Wie in der Abbildung zuvor ist in der Abbildung 3.2 die x-Achse die prozentuale Auslastung des Prozessors und die y-Achse die prozentuale Energieersparnis. Abbildung 3.2 zeigt denselben Trend wie Abbildung 3.1 zuvor, wenn auch nur weniger deutlich.

Im ersten Experiment wurde festgestellt, dass der holistische Scheduler selbst im Power-Saving Modus keine Einbusse in der Performance hat. Dies erlaubt einen Vergleich zwischen dem Edgetic-Scheduler im Power-Saving Modus und dem Default-Scheduler im Performance Modus.

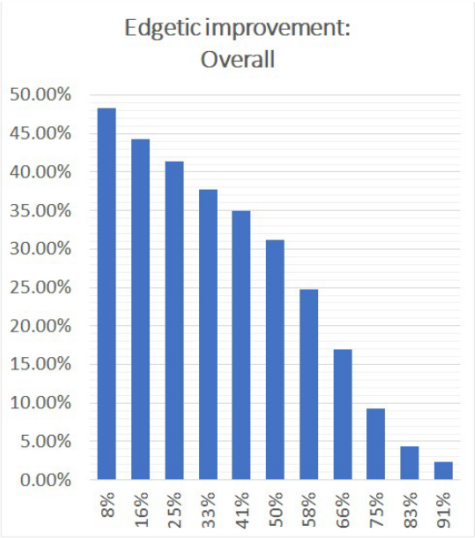


Abbildung 3.3: Energieersparnis vom Edgetic Power-Saving Modus im Vergleich mit dem Default-Scheduler im Performance Modus [3]

In der Abbildung 3.3 ist die y-Achse die prozentuale Energieersparnis des holistischen Schedulers verglichen mit dem Standard-Scheduler. Die x-Achse stellt wiederum die prozentuale Prozessorauslastung dar. Auch in der Abbildung 3.3 zeigt sich der Trend aus den Abbildungen 3.1 und 3.2. Durch den Vergleich eines Power-Saving Modus und eines Performance Modus ist die Energieersparnis viel grösser. Dies bestätigt der Vergleich von Abbildung 3.3 und Abbildung 3.1.

Das zweite Experiment wurde auf einem Cluster mit 215 Servern durchgeführt. Ausserdem wurden zunehmend Workloads verwendet, die vorher nicht bekannt waren, was somit näher an einem realen Szenario ist.

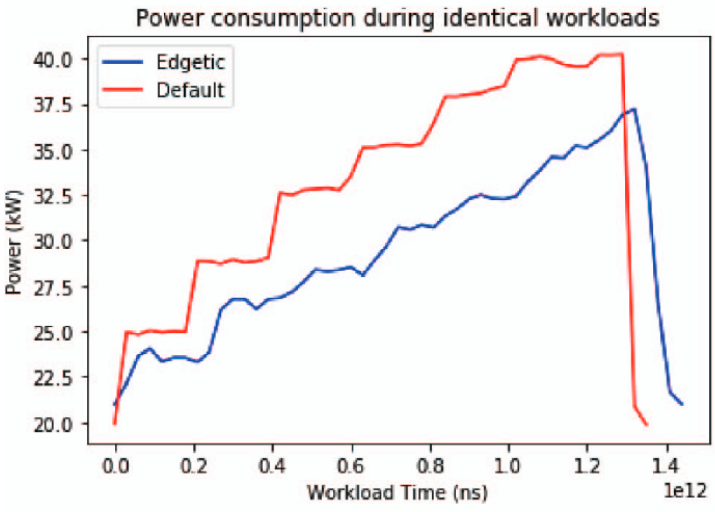


Abbildung 3.4: Energieersparnis von Edgetic Power Saving Mode im Vergleich mit dem Default-Scheduler im Performance Mode [3]

Abbildung 3.4 zeigt den Unterschied im Energieverbrauch auf der y-Achse zwischen dem Standard-Scheduler in Rot und dem Edgetic-Scheduler in Blau. Aus ihr geht deutlich hervor, dass der Edgetic-Scheduler weniger Energie für denselben Workload verbraucht, jedoch minim mehr Zeit benötigt, was auf der x-Achse erkennbar ist. Bei diesem Experiment konnte eine Energieersparnis von 10–20% gemessen werden.

3.2 A Low Carbon Kubernetes Scheduler

Wie in der zuvor vorgestellten Arbeit wird in der Publikation von James und Schien[21] der Standard-Scheduler von Kubernetes ersetzt. Es wird jedoch nicht wie zuvor versucht, die Energieemissionen zu verringern, sondern die CO_2-Emissionen zu senken. Darin werden vier hauptsächliche Vorgehensweisen erklärt, welche es ermöglichen, den CO_2-Ausstoss zu verringern: *Energy efficiency*, *time of use*, *demand response* und *spinning reserve*.

Mit *energy efficiency* ist gemeint, dass durch effizientere Nutzung der Hardware Energie und somit CO_2-Emissionen eingespart werden. Ein Beispiel dafür wäre, dass Hardware-Komponenten je nach momentanem Gebrauch dynamisch ab- und angeschaltet werden.

Time of use beschreibt die Nutzungszeit der Energie. Normalerweise entspricht die Energienutzung der Kurve in Abbildung 3.5. Sie hat einen Höhepunkt, an dem am meisten Energie verwendet wird. Erneuerbare Energien stehen nicht jederzeit zur Verfügung. Nicht dringliche Aufgaben können deshalb zu einem Zeitpunkt ausgeführt werden, an dem mehr erneuerbare Energien zur Verfügung stehen oder weniger davon verbraucht werden.

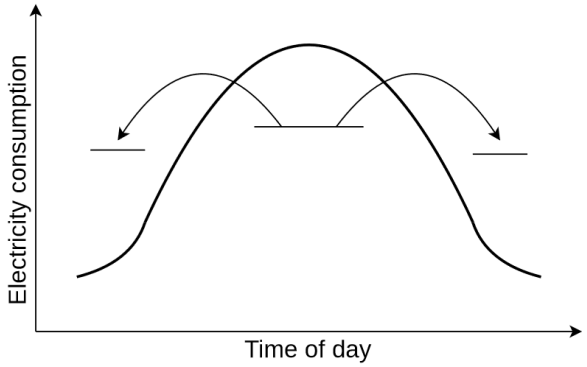


Abbildung 3.5: Energienutzung während eines Tages [21]

Demand response ist die Reduktion der Energienachfrage, einerseits durch direkte Kontrolle der Endgeräte, andererseits durch die Nutzung von Energietarifen. Der Nutzer wird so forciert beziehungsweise motiviert, die Endgeräte sparsamer zu nutzen.

Spinning reserve bezeichnet die Fähigkeit von gewissen Geräten, ihren Energiekonsum an die Belastung des Stromnetzes anzupassen.

In nationalen Stromnetzen wird in der Regel die Energie aus einer Mischung alternativer Quellen erzeugt. Es gibt einen Kohlenstoffkoeffizient, der den durchschnittlichen Ausstoss von Treibhausgasen für die Erzeugung dieser Energie bezeichnet. Dieser Koeffizient variiert von Region zu Region.

Beim Konzept des kohlenstoffarmen Scheduler wird die grünste Region mit einem Datenzentrum ausgewählt. Dann wird berechnet, ob durch die Auslagerung eines Workloads in das grünste Datenzentrum auch tatsächlich der Treibhausgasausstoss reduziert werden kann. Denn nicht nur das Ausführen des Workloads benötigt Energie, sondern auch der zusätzliche Transport der Daten und das Deployment im grünsten Datenzentrum.

Um diesen Scheduler zu testen, wurde in einer Zeitspanne von ungefähr 2 Monaten ein Ranking aller Regionen nach deren Kohlenstoffkoeffizient im Stromnetz durchgeführt. Am besten schnitten in diesem Ranking jedoch nur Länder in Zentraleuropa ab. Deshalb wurde ein neuer Ansatz für den Scheduler verwendet, nämlich die beste Region nach der grössten Sonneneinstrahlung auszuwählen.

3.2.1 Resultate

In der Arbeit von James und Schien[21] wurden drei Versuchsexperimente (`test0`, `test1` und `test2`) durchgeführt, welche als *proof of concept* dienen sollen.

DHI steht in den folgenden Abbildungen für Diffuse Horizontal Irradiance, also die Menge an indirekter Einstrahlung, welche eine horizontale Fläche von der Sonne erhält.

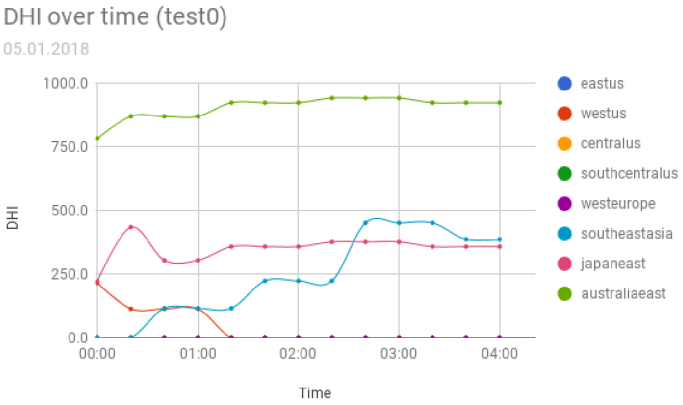


Abbildung 3.6: DHI über Zeit nach Regionen (`test0`) [21]

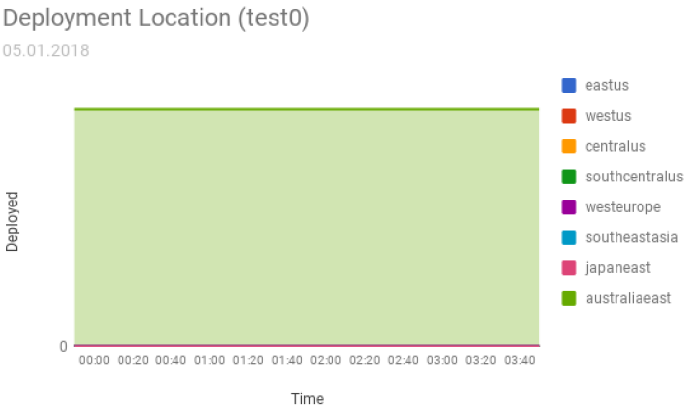


Abbildung 3.7: Regionswahl des Schedulers über Zeit (`test0`) [21]

In Abbildung 3.6 zeigt die y-Achse die Sonneneinstrahlung in Watt pro Quadratmeter. Die x-Achse zeigt die Zeitspanne.

In Abbildung 3.7 zeigt die y-Achse die Region, welche der Low-Carbon-Scheduler ausgewählt hat. Die x-Achse wiederum zeigt die Zeitspanne.

Man kann erkennen, dass in Abbildung 3.6 Ostaustralien mit Abstand die höchste Sonneneinstrahlung hat. Der Low-Carbon-Scheduler wählt deshalb Ostaustralien korrekt als seine Deploymentregion, siehe Abbildung 3.7.

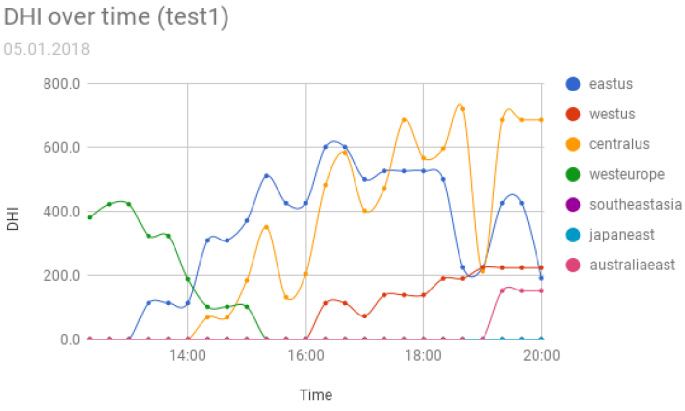


Abbildung 3.8: DHI über Zeit nach Regionen (`test1`) [21]

Abbildung 3.9: Regionswahl des Schedulers über Zeit (`test1`) [21]

In Abbildung 3.8 zeigt die y-Achse die Sonneneinstrahlung in Watt pro Quadratmeter. Die x-Achse zeigt die Zeitspanne.

In Abbildung 3.9 zeigt die y-Achse die Region, welche der Low-Carbon-Scheduler ausgewählt hat. Die x-Achse wiederum zeigt die Zeitspanne.

Von 12 Uhr bis kurz nach 14 Uhr hat Westeuropa in Abbildung 3.8 die höchste Sonneneinstrahlung. Danach bis circa 17:30 Ostamerika und anschliessend bis 20 Uhr Zentralamerika. In Abbildung 3.9 erkennt man, dass in diesen Zeiträumen der Scheduler korrekt zuerst Westeuropa, dann Ostamerika und zuletzt Zentralamerika auswählt.

Im Dritten Versuch (`test2`) wurde der Scheduler ein wenig modifiziert, um die Regionswahl basierend auf der Windstärke zu treffen. Dies soll die Erweiterbarkeit des Schedulers zeigen.

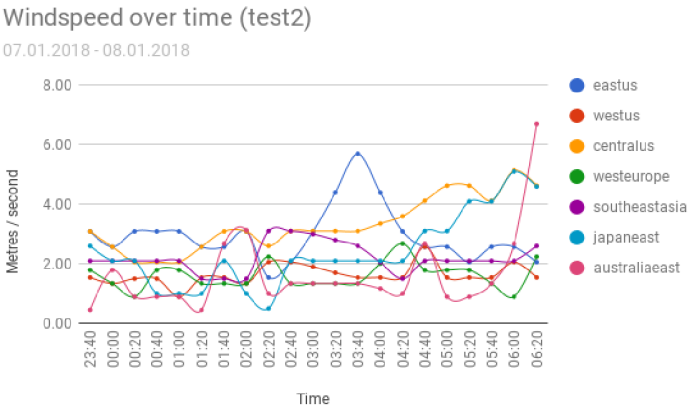


Abbildung 3.10: Windgeschwindigkeit über Zeit nach Regionen (`test2`) [21]

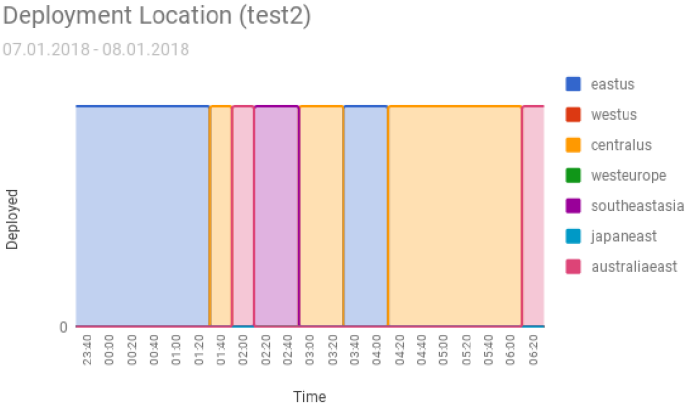


Abbildung 3.11: Regionswahl des Schedulers über Zeit (`test2`) [21]

In Abbildung 3.10 zeigt die y-Achse die Windgeschwindigkeit in Metern pro Sekunde. Die x-Achse zeigt die Zeitspanne.

In Abbildung 3.11 zeigt die y-Achse die Region, welche der Low-Carbon-Scheduler ausgewählt hat. Die x-Achse wiederum zeigt die Zeitspanne.

Anmerkung: Beim Vergleich von Abbildung 3.10 und 3.11 ist aufgefallen, dass der Scheduler die Regionen nicht ganz korrekt auswählt. In Abbildung 3.10 beispielsweise ist die Ost-USA von ca. 03:00 bis 4:10 die Region, mit der höchsten Windgeschwindigkeit. Jedoch in Abbildung 3.11 wurde die Region Ost-USA erst ab 03:30 als beste Region ausgewählt. Im Paper wird jedoch darauf nicht eingegangen.

3.3 Auswertung der Recherche

Der holistische Scheduler[3] ist in der Lage, mithilfe eines Prognosemodells Energie einzusparen. Für die SIX wäre es daher möglich und interessant, einen solchen Ansatz zu implementieren. Der Fokus dieser Arbeit liegt jedoch darauf zu untersuchen, wo und welches die grössten Einflussfaktoren für die Energiereduktion sind. Dazu werden Ansätze betrachtet, die den Energieverbrauch im Detail analysieren. Der holistische Scheduler, der ein Cluster als Ganzes betrachtet, wird deshalb in dieser Arbeit nicht weiterverfolgt.

Der kohlenstoffarme Scheduler[21] ist ein theoretischer Ansatz, der erst teilweise in die Praxis umgesetzt wurde. Dieser Scheduler verschiebt Workloads in die Region, in welcher derzeit der nachhaltigste Strom verfügbar ist. Diese Entscheidung basiert auf Einflussfaktoren wie der Sonneneinstrahlung und der Windstärke. Dadurch würden Workloads auch potenziell in Regionen ausserhalb der Schweiz verschoben werden. Für die SIX, welche Aufgrund von verschiedenen Auflagen an den Datenstandort Schweiz gebunden ist, macht es momentan deshalb keinen Sinn, diesen Ansatz weiterzuverfolgen. Die Auslagerung von Workloads und damit von Daten könnte auch zu datenschutzrechtlichen Problemen führen. Daher wird auch dieser Ansatz in dieser Arbeit nicht weiter verfolgt.

4 Hauptteil

Im Kapitel 4.1 wird die Installation von Kepler beschrieben. Anschliessend wird in den Kapitel 4.2 und 4.3 auf die beiden Ansätze zur Optimierung der Energieeffizienz eingegangen, welche Antworten auf die Forschungsfragen geben sollen.

4.1 Installation Kepler

Für die Erfassung des Energieverbrauchs war es notwendig, Kepler auf einem Cluster zu installieren. Beim verwendeten Test-Cluster handelt es sich um ein Non-Prod Cluster der SIX bestehend aus zwei Master- und 22 Worker-Nodes auf welchen ca. 5000 Pods laufen.

Zur Installation von Kepler stehen verschiedene Methoden wie die Nutzung eines Operators, eines Helm-Charts oder die Nutzung von Manifest-Skripts zur Verfügung[18]. Im Rahmen dieser Arbeit stellte sich heraus, dass die Installation mittels eines angepassten Helm-Charts die einzig funktionierende Option war. Um eine isolierte Umgebung für den Betrieb von Kepler zu gewährleisten wurde die Installation in einem dedizierten Namespace durchgeführt. Aufgrund von Instabilitäten im internen Monitoring-Stack des Test-Clusters war zusätzlich noch die Installation einer eigenen Prometheus und Grafana-Instanz im selben Namespace notwendig.

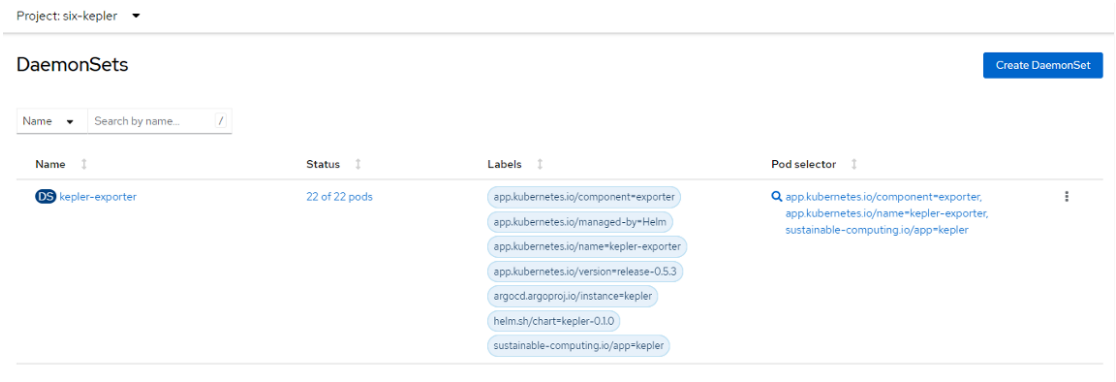


Abbildung 4.1: Installation von Kepler als DaemonSet im Test-Cluster

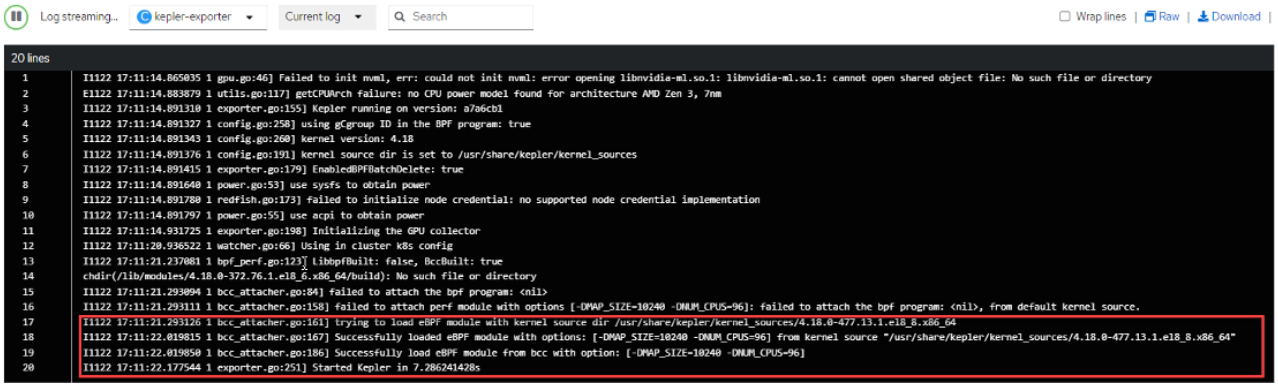


Abbildung 4.2: Einblick in die Logs von Kepler: Aktivierung von eBPF sowie ACPI/Redfish

Abbildung 4.1 zeigt die Installation von Kepler im Test-Cluster. Der `kepler-exporter` (siehe Kapitel 2.2) wurde als DaemonSet (siehe Kapitel 2.1.2) installiert, um eine umfassende Überwachung über alle Nodes des Clusters zu gewährleisten.

Für detailliertere Einblicke in die Energieverbrauchsdaten wurden eBPF und die Nutzung der Redfish-API aktiviert. Abbildung 4.2 zeigt die erfolgreiche Aktivierung in den Logs von Kepler.

4.2 Ansatz 1: Monolith vs. Microservices

Um auf die erste Forschungsfrage bezüglich «Auf welcher Ebene (Deploymentstrategie/Architektur) hat man die grössten Einflussfaktoren in Bezug auf Energieersparnis bei gleichbleibender Performance?» einzugehen, ist zuerst wichtig zu verstehen, was damit gemeint ist.

Unter *Deploymentstrategie/Architektur* wird die verwendete Strategie zum Deployen der Applikation resp. der Architekturansatz verstanden. Bekannte Architekturansätze welche auch im Cloud Computing verwendet werden, sind unter anderem *Monolithen* und *Microservices.*

Monolithen sind grosse, einzelständige Applikationen und Microservices hingegen mehrere kleinere, voneinander Abhängige. Deshalb spricht man auch von Deploymentstrategie, da man beim Deployment eines Monolithen ein Deployment einer grossen, ressourcenintensiven Applikation hat, beim Deployment von Microservices hingegen werden mehrere Applikationen mit kleiner Ressourcenlast deployt.

Mithilfe eines Experiments wurde deshalb der Unterschied des Energieverbrauchs zwischen eines Monoliths und einer auf Microservices basierender Applikation geprüft. Damit sollte die Effizienz der Ressourcennutzung in Kubernetes-Clustern bei variierender Pod-Anzahl und CPU-Zuweisung evaluiert werden und somit Aufschluss auf die erste Forschungsfrage bezüglich der Ebene mit den grössten Einflussfaktoren geben.

4.2.1 Aufbau des Experiments

Im ersten Experiment wurden zwei unterschiedliche Deployment-Szenarien verglichen:

- ein Deployment mit einem einzigen Pod, welcher einen hohen CPU-Wert zugewiesen bekam
- ein Deployment mit mehreren Pods, bei denen jeder einen geringen CPU-Wert erhielt

Das Tool `stress-ng` simulierte dabei mittels eines vorhersehbaren Workloads die Auslastung. Dies ermöglichte die Untersuchung, wie die unterschiedlichen CPU-Werte den Energieverbrauch beeinflussen.

4.2.2 Ablauf und Konfiguration

Bei diesem Experiment wurde jeweils ein Helm-Chart für die beiden unterschiedlichen Deployment-Szenarien erstellt. Das Values-File der Charts wurde so konfiguriert, dass die CPU-Zuweisung für die Pods den Anforderungen des Experiments entspricht.

Dies wurde einerseits durch das Setzen der Ressourcenanforderungen (`requests`) sichergestellt, andererseits durch die feste Konfiguration der generierten CPU-Last mittels `stress-ng`.

Die Gewährleistung einer möglichst konsistenten Umgebung erforderte, dass beide Deployments auf dem gleichen Node `worker-node1` deployt wurden. Um zusätzliche Isolation zu sicherzustellen, wurden die Deployments in separate Namespaces durchgeführt. Diese Namespaces wurden jeweils zusammen mit den Deployments erstellt.

Die vollständigen Helm Charts finden sich unter *kepler-test-szenarios* im Gitlab-Repository der Bachelorarbeit.

Listing 1 zeigt einen Auszug der relevanten Abschnitte der gerenderten Konfiguration des Helm-Charts für das Deployment mit einem Pod und 8 CPUs:

Listing 1: Auszug aus Helm-Chart Konfiguration für 1 Pod mit 8 CPUs

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: stress-ng-single
  [...]
spec:
  replicas: 1
  [...]
  template:
    [...]
    spec:
      containers:
        - image: "kubestress-docker-remote/alexeiled/stress-ng:latest-ubuntu"
          command:
              - "stress-ng"
              - "-cpu"
              - "8"
              - "--cpu-load"
              - "100"
              - "--timeout"
              - "0"
          resources:
            requests:
              cpu: "8"
      nodeName: "worker-node1"
```

Listing 2 zeigt einen Auszug der relevanten Abschnitte der gerenderten Konfiguration des Helm-Charts für das Deployment mit 8 Pods und je 1 CPU:

Listing 2: Auszug aus Helm-Chart Konfiguration für 8 Pods mit 1 CPUs

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: stress-ng-multiple
  [...]
spec:
  replicas: 8
  [...]
  template:
    [...]
    spec:
      containers:
        - image: "kubestress-docker-remote/alexeiled/stress-ng:latest-ubuntu"
          command:
              - "stress-ng"
              - "-cpu"
              - "1"
              - "--cpu-load"
              - "100"
              - "--timeout"
              - "0"
          resources:
            requests:
              cpu: "1"
      nodeName: "worker-node1"
```

4.2.3 Auswertung

Beide Deployment-Szenarien liefen vom 20.12.2023 von 14:00 für 24 Stunden und wurden anschliessend ausgewertet.

Dazu wurde jeweils die durch den `kepler-exporter` zur Verfügung gestellte Prometheus-Metrik `kepler_container_joules_total` der beiden Szenarien verglichen und gegenübergestellt. Bei dieser Metrik handelt es sich um den aggregierten Energieverbrauch aller verfügbaren Komponenten für einen bestimmten Container eines Pods[17].

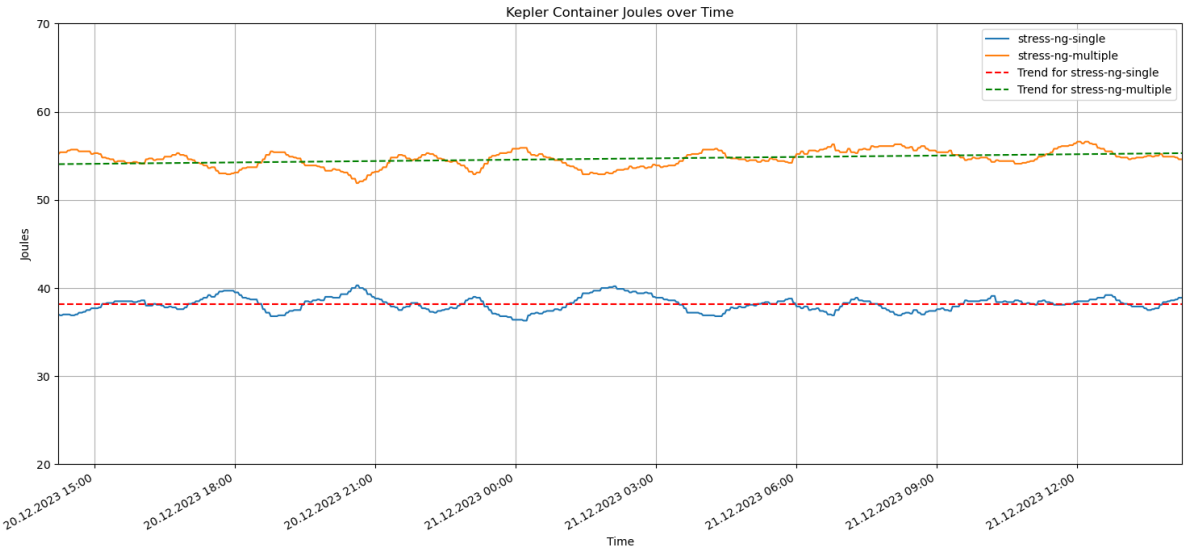


Abbildung 4.3: Gegenüberstellung beider Deployment-Szenarien

Abbildung 4.3 zeigt den Energieverbrauch ab 20 Joules auf der y-Achse über die Zeit auf der x-Achse. Es ist erkennbar, dass sich die Werte für das Szenario mit nur einem, dafür grösseren Pod (`stress-ng-single`) um die 40 Joules bewegen, und diese für das Szenario der mehreren, kleinen Pods (`stress-ng-multiple`) um die 55 Joules.

Die Auswertung der Ergebnisse dieses Experiments offenbarte Schwierigkeiten bei der Bestimmung der Zusammenhänge zwischen dem Energieverbrauch und den CPU-Werten über die verschiedenen Pods hinweg. Obwohl ersichtlich ist, dass sich die Werte für das Szenario mit dem einen grösseren Pod in einem kleineren Wertebereich bewegen, ist nicht klar, ob dies wirklich mit den konkreten CPU-Werten zusammenhing oder ob gegebenenfalls die kleineren Pods einfach generell einen Management-Overhead hatten, welcher auch zu mehr Energieverbrauch führte.

Eine direkte Auswirkung der CPU-Zuteilung auf den Energieverbrauch war jedenfalls nicht eindeutig bestimmbar. Um diesen Zusammenhang zu erschliessen, wurde das Experiment so angepasst, dass der Energieverbrauch eines einzelnen Pods zusammen mit einem fest definierten CPU-Wert isoliert betrachtet werden konnte.

Das Setup des Experiments wurde so modifiziert, dass eine Reihe von Pods durch verschiedene Deployments innerhalb eines einzelnen Namespaces auf dem gleichen Node deployt wurden. Dabei wurde jedem Pod ein unterschiedlicher, aber fixer CPU-Wert zugewiesen. In den Deployments wurde jeweils ein Pod gestartet, auf dem ein Lasttest mit `stress-ng` im Bereich dieses Fixwertes durchgeführt wurde. Für das Experiment wurden Fixwerte von 1 bis 7 CPUs gewählt.

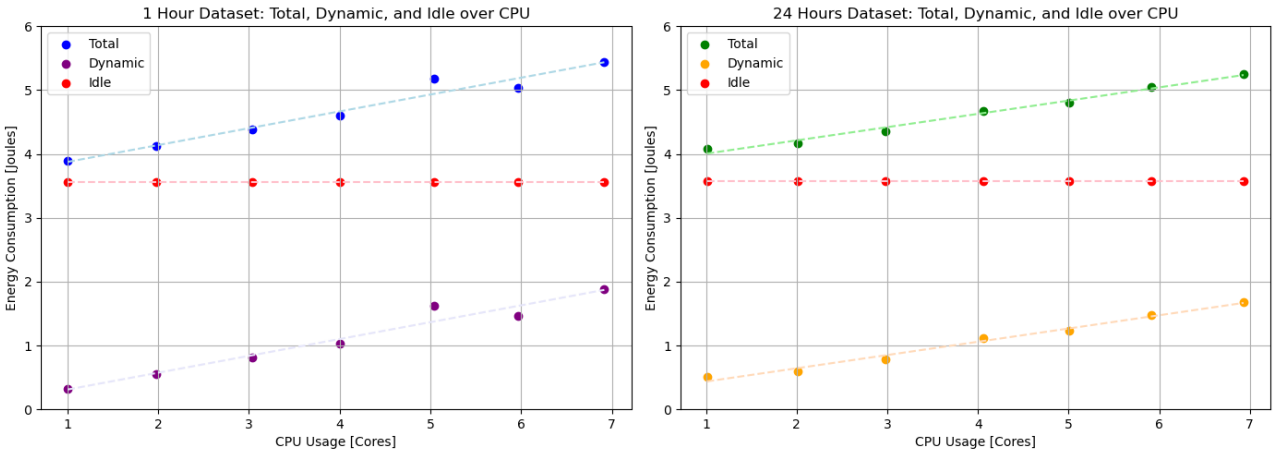
Listing 3 zeigt einen Auszug der relevanten Abschnitte der gerenderten Konfiguration des Helm-Charts für ein solches Deployment mit einem Pod und n CPU, wobei n für den konkreten CPU-Wert steht, welcher dem Pod zugewiesen wurde:

Listing 3: Auszug aus Helm-Chart Konfiguration für 1 Pods mit unterschiedlichen CPU-Werte

```
kind: Deployment
[...]
spec:
  replicas: 1
  [...]
  template:
    [...]
    spec:
      containers:
        - image: "kubestress-docker-remote/alexeiled/stress-ng:latest-ubuntu"
          command:
              - "stress-ng"
              - "-cpu"
              - "n"
              [ identische werte wie in den anderen listings ]
          resources:
            requests:
              cpu: "n"
      nodeName: "worker-node1"
```

Dieses Experiment wurde einmal über eine Stunde und einmal über 24 Stunden ausgewertet. Neben den Metriken von Kepler wurde diesmal auch die CPU-Load erfasst, um den Energieverbrauch in Relation zur CPU-Auslastung zu setzen.

Die Auswertung führte zur Feststellung, dass es zwei verschiedene Modi des durch Kepler erfassten Energieverbrauchs gibt (*dynamic* und *idle*, siehe auch [16]). Der dynamische Energieverbrauch hängt dabei von der Ressourcennutzung ab. Der Idle Energieverbrauch ist die konstante Leistung, die sich nicht ändert, unabhängig davon, ob sich das System im Ruhezustand oder unter Last befindet. Der Idle Energieverbrauch hat dabei nichts mit der Idle Time einer CPU zu tun. Die Idle Time ist die Zeit, in welcher ein Prozess wartet, bis er vom CPU wieder aufgerufen wird. Der Idle Energieverbrauch ist hingegen der Grundenergieverbrauch/Overhead eines Nodes.



CPU	Dynamic	Idle	Total
1.01	0.33	3.57	3.89
1.98	0.56	3.57	4.12
3.04	0.81	3.57	4.38
4.00	1.03	3.57	4.60
5.04	1.62	3.57	5.19
5.96	1.47	3.57	5.04
6.92	1.88	3.57	5.45

CPU	Dynamic	Idle	Total
1.01	0.51	3.57	4.08
2.01	0.60	3.57	4.17
2.98	0.79	3.57	4.36
4.06	1.11	3.57	4.68
5.01	1.23	3.57	4.80
5.91	1.48	3.57	5.05
6.93	1.68	3.57	5.25

Abbildung 4.4: Auswertung der Total, Dynamic und Idle über CPU Usage

In Abbildung 4.4 wurden die verschiedenen CPU-Fixwerte auf der x-Achse dem Energieverbrauch in Joules auf der y-Achse gegenübergestellt. Dabei wurde der Energieverbrauch in Idle, Dynamic und Total aufgeteilt.

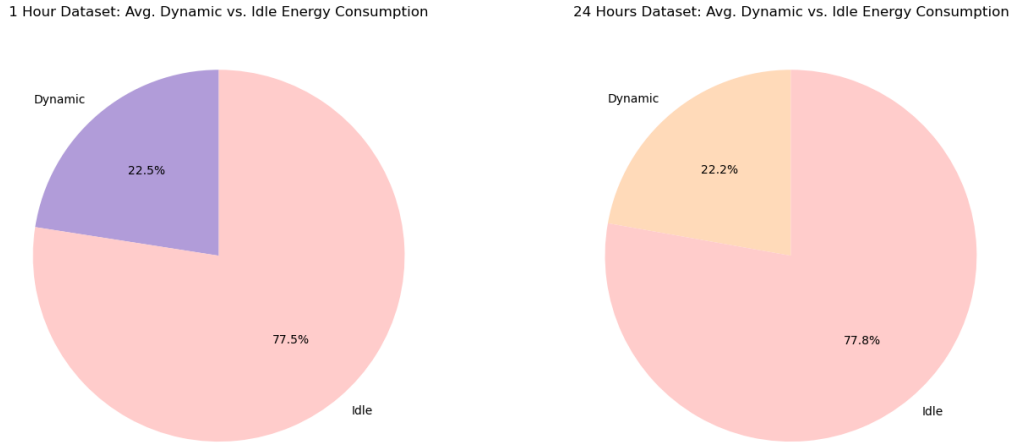


Abbildung 4.5: Average Ratio von Idle vs. Dynamic

Abbildung 4.5 zeigt das durchschnittliche Verhältnis von Dynamic versus Idle des Experiments.

Durch dieses angepasste Experiment konnten mehrere Zusammenhänge festgestellt werden:

- Abbildung 4.4 zeigt einen linearen Zusammenhang zwischen CPU- und Energieverbrauch.
- Gemäss Abbildung 4.4 ist der Idle-Energieverbrauch unabhängig von der CPU Usage und über jeden gemessenen CPU-Wert identisch. Dies ist darauf zurückzuführen, dass die Pods auf demselben Node liefen.
- In Abbildung 4.5 ist sichtbar, dass der idle-Wert im Vergleich zum dynamic-Wert einen viel grösseren Anteil ausmacht.

Gemessen am dynamischen Energieverbrauch verbraucht ein Pod mit sieben CPUs 20% weniger Energie als sieben Pods mit jeweils einer CPU. Da der HPA (wie in Kapitel 2.1.4 erläutert) die Anzahl der Pods steuert, ergibt es im Hinblick auf den nächsten Ansatz (*Horizontal Pod Autoscaler* resp. *Vertical Pod Autoscaler*), keinen Sinn, den Horizontal Pod Autoscaler zu betrachten.

Allerdings wurde mit stress-ng immer eine konstante CPU-Last erzeugt, was natürlich nicht dem Real-World-Szenario entspricht. In der realen Welt haben Applikationen typischerweise kurzzeitige Spitzen in der CPU-Auslastung und dann wieder Phasen mit geringerer Auslastung. Ein Beispiel hierfür sind Trading-Applikationen, die während der Öffnungszeiten der Börse am Morgen bis zur Schliessung am Abend hohe CPU-Auslastung hat, aber in der Nacht eine geringere. In diesem Fall wäre der Ansatz des HPA interessant, da man damit die Auslastung über den Tag hinweg durch Hinzufügen von Pods regulieren und diese am Abend wieder freigeben könnte.

In der Applikationslandschaft der SIX gibt es historisch bedingt viele Monolithen und die Cloud Native Transformation schreitet nur langsam voran. Die Herausforderung liegt insbesondere darin, die bestehenden Applikationen so umzugestalten, dass sie eine saubere horizontale Skalierung ermöglichen. Daher ist die vertikale Skalierung energietechnisch durchaus sinnvoll.

Der dynamische Teil des Energieverbrauchs ist direkt an CPU Usage gekoppelt. Optimierungen betreffend der CPU Usage liegt in der Verantwortung der Applikation. Der Idle-Verbrauch pro Pod, welcher einen weitaus höheren Anteil des Energieverbrauchs darstellt, ist auf die schlechte Auslastung des Nodes zurückzuführen. Dies zeigt eine Erfassung des Idle-Wertes über alle Pods aller Nodes (siehe Abbildung 4.6).

Um den Idle-Anteil zu senken, müssen die Pods auf weniger Nodes verteilt werden. Das Entfernen von Nodes bringt jedoch alleine nichts, da Pods immer noch gleich viel CPU in ihren `requests` anfordern und somit viele Pods dann einfach nicht gescheduled werden (siehe Kapitel 2.1.4).

Ein besserer Ansatz wäre, durch ein optimaleres *sizing* der Pods eine höhere Anzahl an Pods auf die Nodes zu erreichen, um somit eine bessere Auslastung zu erzielen. Dies ist am besten mit einem *Vertical Pod Autoscaler* (VPA) zu erreichen, da dieser unter anderem die `requests` der Pods anhand von berechneten `recommendations`, welche auf der effektiven Nutzung basieren, besser optimieren kann. Damit ist es dann möglich, Nodes zu entfernen und dadurch die Idle-Werte zu verbessern.

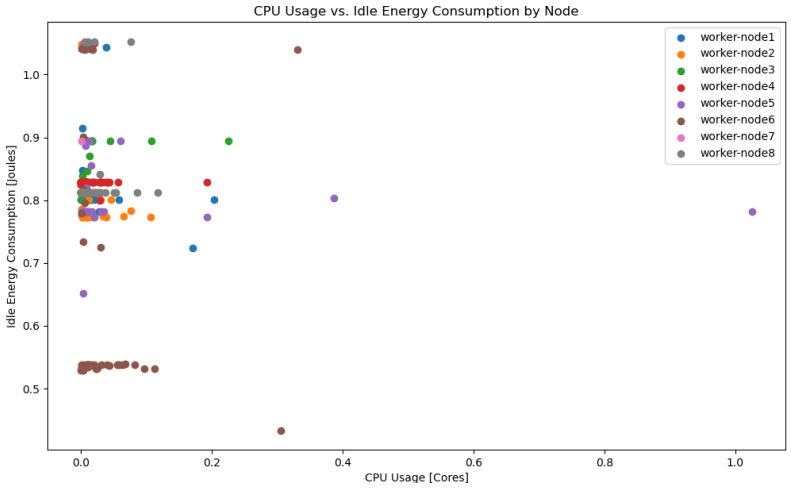


Abbildung 4.6: CPU Usage vs. Idle Energieverbrauch pro Node

Abbildung 4.6 stellt die Erfassung des Idle-Wertes über alle Pods aller Nodes dar. Die x-Achse zeigt die CPU-Auslastung in Cores. Auf der y-Achse ist der Idle-Energieverbrauch in Joule abgebildet. Aus den durch die Punkte formierten Linien innerhalb des Plots geht klar hervor, dass CPU Usage keinen Einfluss auf den Idle Energieverbrauch hat. Auch ist ersichtlich, dass alle Pods, welche auf den gleichen Nodes ausgeführt werden, identische Werte zum Idle Energieverbrauch besitzen.

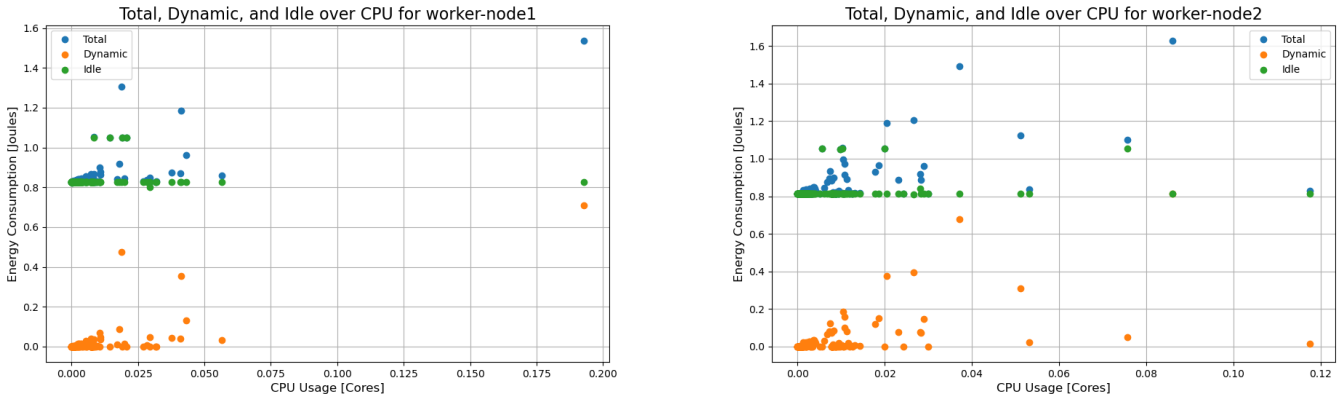


Abbildung 4.7: Usage Analysis für worker-node1 und worker-node2 (exemplarisch)

In Abbildung 4.7 wurde exemplarisch für `worker-node1` und `worker-node2` die gleichen Power-Usage-Metriken ausgewertet wie in Abbildung 4.4, nämlich Total, Dynamic und Idle. Die Auswertung verbildlicht, dass der Dynamic Energieverbrauch (orange) verschwindend klein ist und der Idle-Anteil (grün) verhältnismässig hoch.

4.3 Ansatz 2: Vertical Pod Autoscaler

Der erste Ansatz aus Kapitel 4.2 zeigte auf, dass der Idle Energieverbrauch das grösste Potenzial für Optimierungen bietet. Basierend auf diesen Erkenntnissen wurde der Einsatz des Vertical Pod Autoscalers weiterverfolgt. Der Einsatz des VPA zielte darauf ab, die Pod-Grössen dynamisch anzupassen, um deren Ressourcennutzung zu optimieren. So soll eine effizientere Pod-Verteilung auf den Nodes erreicht werden. Dadurch soll es möglich werden, Nodes aus dem Cluster entfernen zu können, was den Idle Energieverbrauch deutlich verringern sollte.

4.3.1 Ablauf und Konfiguration

Als Vorbereitung für das Setup wurde der VPA als Operator von Red Hat[22] gemäss Instruktionen[23] über den Operator-Hub auf den Test-Cluster installiert.

Um diesen zu nutzen, bedarf es jeweils einer speziellen Custom Resource (CR) pro Workload. Diese CR assoziiert den Workload mit einem Modus des VPAs.

Listing 4: Beispielskonfiguration einer solchen VPA CR

```
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: my-app-vpa
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind:       Deployment
    name:       my-app
  updatePolicy:
    updateMode: "Auto"
```

Wie im obigen Listing 4 ersichtlich ist, besteht ein solches CR aus einer Referenz zum Workload (`targetRef`) und einem `updateMode`, welcher einer der nachfolgenden Werte annimmt[24]:

1. *Auto* — Derzeit entspricht dies der Funktion *Recreate*.
2. *Recreate* — Der VPA weist die Ressourcenanforderungen bei der Pod-Erstellung zu und aktualisiert sie bei bestehenden Pods. Dies erreicht der VPA, indem er sie *evicted*, falls die angeforderten Ressourcen erheblich von der neuen Empfehlung abweichen (unter Berücksichtigung des Pod Disruption Budget, falls definiert).
3. *Initial* — Der VPA weist die Ressourcenanforderungen nur bei der Pod-Erstellung zu und wird sie aber später nie mehr ändern.
4. *Off* — Der VPA ändert die Ressourcenanforderungen der Pods nicht automatisch. Die Empfehlungen werden aber berechnet und können im VPA-Objekt eingesehen werden.

Für das Setup des Experiments wurde ein rudimentärer Operator geschrieben, welcher solche CRs für alle Deployments im Cluster erstellte, die `targetRef` korrekt setzte und den `updateMode` je nach Testphase wahlweise auf *Auto* oder *Off* schaltete.

4.3.2 Phase 1: Verhalten mit VPA

In der ersten Phase des Experiments wurde der `updateMode` auf *Auto* und somit implizit auf *Recreate* gesetzt, um bei der Erstellung der Pods die Ressourcenanforderungen auf empfohlene Werte des VPAs zu setzen. Zeitgleich wurden zwei Nodes als *unschedulable* markiert, um eine Verteilung der Pods auf andere Nodes zu erzwingen. Anschliessend wurden sämtliche Pods aller Namespaces auf dem Cluster neu gestartet, um die neuen Ressourcenanforderungen sowie die

Umverteilung auf weniger Nodes zu erzwingen und einen einheitlichen Messbeginn-Zeitpunkt zu erhalten.

Dieses Setup wurde so für 24 Stunden laufen gelassen, um eine Datenlage über die Auswirkungen des VPAs auf die Ressourcennutzung und den Energieverbrauch zu evaluieren. Dabei wurden folgende Werte überwacht: Dynamic Energieverbrauch, Idle Energieverbrauch, CPU-Load, also die gleichen Werte wie im ersten Ansatz, sowie zusätzlich die Pod CPU requests und limits.

4.3.3 Phase 2: Verhalten ohne VPA

Für die zweite Phase wurde der `updateMode` auf *Off* gesetzt, um damit den Effekt des VPAs wieder rückgängig zu machen und die Ressourcenanforderungen der Pods wieder auf ihre ursprünglichen Werte zu setzen. Die beiden im vorherigen Kapitel 4.3.2 als *unschedulable* markierten Nodes wurden wieder auf *schedulable* gesetzt und nahmen wieder Pods auf. Anschliessend wurden wiederum alle Pods neu gestartet, damit sie die ursprünglichen Ressourcenanforderungen bekamen und wieder auf alle vorhandenen Nodes verteilt wurden.

Dieses Setup wurde ebenfalls während 24 Stunden laufen gelassen, um die ursprüngliche, respektive momentane Sicht auf die Ressourcennutzung und den Energieverbrauch zu bekommen. Es wurden dieselben Werte wie in der ersten Phase überwacht.

4.3.4 Auswertung

Nach den Messungen wurden die gesammelten Daten ausgewertet. Dabei ist wichtig zu erwähnen, dass es sich hierbei um ein bereinigtes Datenset handelt. Es betrachtet nur Pods, welche während der Gesamtzeit der 24 Stunden liefen, also keine kurzlebigen Pods. Ebenfalls wurden gewisse interne Namespaces herausgefiltert, um das Datenset nicht zu verfälschen.

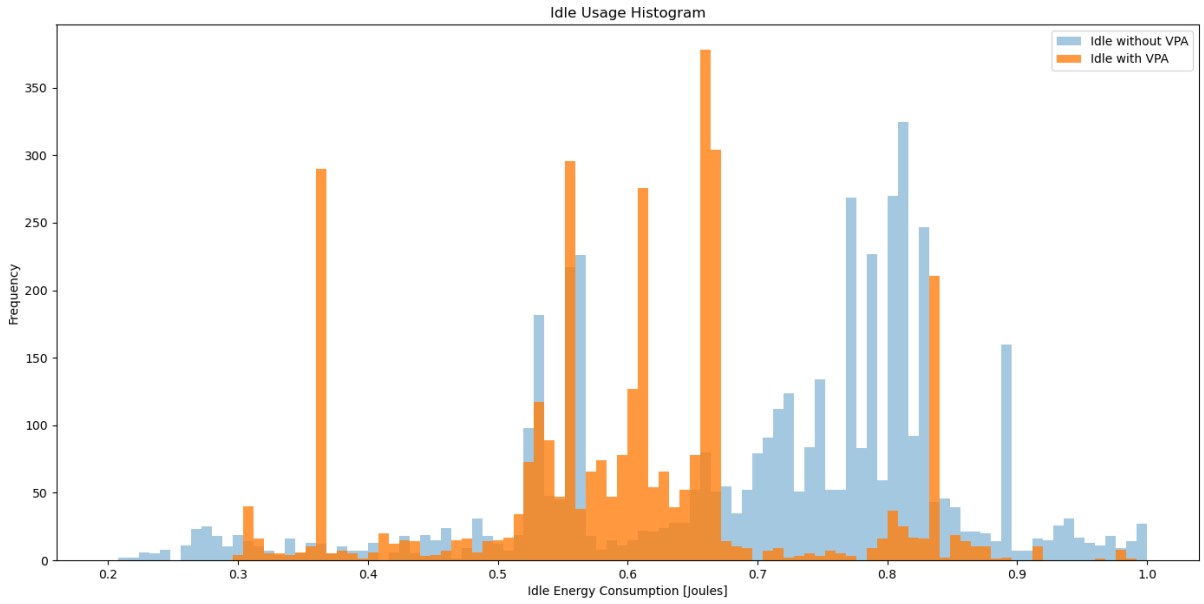


Abbildung 4.8: Idle Energieverbrauch Histogram mit und ohne VPA

Abbildung 4.8 zeigt die Verteilung des Idle Energieverbrauchs für Pods in einem Kubernetes-Cluster mit und ohne VPA. Auf der x-Achse wird der Idle Energieverbrauch der Pods in Joules angegeben. Die y-Achse zeigt die Häufigkeit, mit der die Werte innerhalb der auf der x-Achse angegebenen Bereiche der CPU-Nutzung über alle Pods aller Nodes auftreten.

Aus Abbildung 4.8 geht klar hervor, dass der VPA funktioniert hat. Die Werte des Idle Energieverbrauchs mit VPA ist konsistent tiefer als die ohne VPA. Dies zeigt, dass damit der Idle Energieverbrauch erfolgreich optimiert werden konnte.

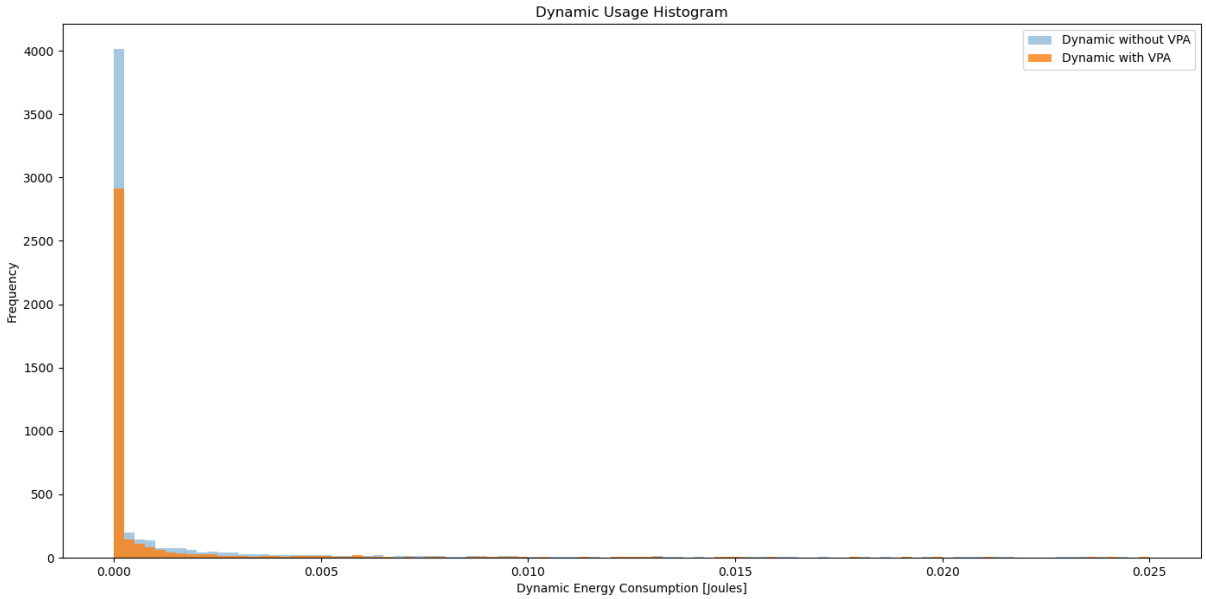


Abbildung 4.9: Dynamic Energieverbrauch Histogram mit und ohne VPA

Abbildung 4.9 zeigt dieselbe Verteilung der CPU-Nutzung für Pods in einem Kubernetes-Cluster mit und ohne VPA wie Abbildung 4.8, nur für den Dynamic Energieverbrauch statt für den Idle Energieverbrauch. Auf der x-Achse ist der Dynamic Energieverbrauch der Pods in Joules angegeben, auf der y-Achse die Häufigkeit, mit der die Werte innerhalb der angegebenen Bereiche der CPU-Nutzung über alle Pods aller Nodes auftreten.

Aus Abbildung 4.9 geht hervor, dass die Werte für den Dynamic Energieverbrauch mit und ohne VPA ähnlich verlaufen. Die etwas kleinere Häufigkeit mit VPA ist darauf zurückzuführen, dass im Zeitraum der Messung mit VPA eine geringere Anzahl Pods über allen Nodes ausgeführt wurde. Anhand der verschwindend kleinen Werte auf der x-Achse ist zudem gut ersichtlich, dass sehr viele kleine Pods auf allen Nodes ausgeführt werden.

Durch die Ermittlung des mittleren Idle Energieverbrauch pro Pod und die mittlere Anzahl der Pods wurde den totalen mittleren Idle Energieverbrauch für alle Pods mit und ohne VPA berechnet. Somit konnte erfolgreich gezeigt werden, dass bei gleichbleibendem dynamischen Energieverbrauch der Idle-Wert durch den Einsatz des VPA um 26.39% Prozent verringert werden konnte.

Es ist wichtig zu erwähnen, dass der Einsatz eines VPA nicht in jeder Situation sinnvoll ist. In einem Cluster mit ressourcenintensiven Java-Applikationen ist der Einsatz nicht zielführend. Eine Java-Applikation hat typischerweise vor allem beim Startvorgang durch das Cache-Warming der JVM einen erhöhten CPU-Verbrauch und anschliessend einen verhältnismässig tieferen. Da der VPA bei seinen Empfehlungen die gesamte Laufzeit berücksichtigt und dabei sowohl Requests als auch Limits anpasst, könnten zu niedrige Werte festgelegt werden. Dies hätte eine Verzögerung des Startvorgangs und ein Versagen der Readiness Checks zur Folge. Im schlimmsten Fall könnten Pods überhaupt nicht mehr gescheduled werden.

5 Diskussion

In diesem Kapitel werden die Resultate aus Kapitel 4 aufgegriffen und im Kapitel 5.1 interpretiert. Dort werden ebenfalls die im Kapitel 1.2 beschriebenen Forschungsfragen beantwortet. Im Kapitel 5.2 wird schlussendlich erörtert, in welche Richtung man weiterforschen könnte.

5.1 Fazit

Die Einrichtung der Umgebung für Kapitel 4 war aufgrund von Schwierigkeiten bei der Installation und Konfiguration des noch sehr jungen Kepler-Projekts und der Instabilität der Monitoring-Tools innerhalb des Test-Clusters eine Herausforderung.

Schlussendlich konnten aber diese Hürden überwunden und die Analyse sowie die Experimente für die unterschiedlichen Ansätze durchgeführt werden. Ziel des ersten Ansatzes aus Kapitel 4.2 war es, die erste Forschungsfrage aus Kapitel 1.2 zu beantworten:

- Auf welcher Ebene (Deploymentstrategie / Architektur) hat man die grössten Einflussfaktoren in Bezug auf Energieersparnis bei gleichbleibender Performance?

Die ersten Resultate des Experimentes zeigten, dass von den beiden Deploymentstrategien Microservices deutlich mehr Energie verbraucht als ein Monolith. Es war jedoch unklar, ob die CPU-Zuteilung einen direkten Einfluss auf den Energieverbrauch hatte. Um dies herauszufinden wurde das Experiment leicht angepasst, sodass der Energieverbrauch eines einzelnen Pods in Verbindung mit einem festgelegten CPU-Wert isoliert untersucht werden konnte. Dabei kam heraus, dass Deploymentstrategie und Architektur nur einen Einfluss auf den dynamischen Energieverbrauch haben. Der mit 77% weitaus grössere Anteil des Idle Energieverbrauchs (siehe Abbildung 4.5) liegt jedoch nicht in der Hand der Applikation, sondern des Plattformproviders.

Im Bezug zur Forschungsfrage bedeutet dies, dass die Ebene der Deploymentstrategie/Architektur nur einen geringen Einflussfaktor im Bezug zur Energieersparnis hat. Diese Erkenntnisse führen zur Beantwortung der zweiten Forschungsfrage aus Kapitel 1.2:

- Welche Faktoren haben aus Sicht einer auf Kubernetes deployten Applikation den meisten Einfluss in Bezug auf die Energieersparnis?

Der grösste Faktor in Bezug auf Energieersparnis aus Sicht einer auf Kubernetes deployten Applikation im Rahmen dieser Arbeit ist der Idle Energieverbrauch der Nodes. Ein weiterer Faktor ist der dynamic-Wert, welcher für die dynamischen Energiekosten der deployten Applikation steht. Dieser hatte im Rahmen dieser Arbeit jedoch nur einen geringen Einfluss auf die Energieersparnis.

Im Ansatz des Kapitels 4.3 wurde Mithilfe eines VPA versucht, den Idle Energieverbrauch der Nodes zu optimieren. Durch den Betrieb des VPA und dem Entfernen der Nodes konnte bei gleichbleibendem dynamischen Energieverbrauch der Idle-Wert um 26.39% verringert werden.

Die Erkenntnisse dieser Arbeit haben innerhalb der SIX viel Diskussionen über den effizienten Betrieb einer Kubernetes-Infrastruktur ausgelöst. Themen wie optimales Cluster Sizing und Ressourceneffizienz werden deshalb auch nach dieser Arbeit weiter verfolgt.

5.2 Ausblick

Im Rahmen dieser Arbeit wurden Möglichkeiten zur Optimierung des Idle Energieverbrauchs angeschaut. Es ist wichtig zu erwähnen, dass nur eine konstante CPU-Auslastung getestet wurde und der VPA keine Lösung für alle Situationen ist. Im Umfeld von vielen Java-Applikationen, welche eine erhöhte Auslastung für das Startup der JVM benötigen, müsste der Recommender des VPA optimiert werden und ein Custom Recommender, welcher Request und Limits separat betrachtet, verwendet werden.

Es empfiehlt sich zudem, den Einsatz eines HPA zur Optimierung des Energieverbrauchs noch genauer zu untersuchen. In dieser Arbeit wurde durch den Einsatz der konstanten CPU-Auslastung jeweils nur das Verhalten von gleichbleibender Auslastung getestet, was natürlich nicht wirklich dem Verhalten in der echten Welt entspricht. Es wäre zu untersuchen, inwiefern sich bei ungleichmässiger Last die Nutzung eines VPA eignen würde.

Weiter wäre zu untersuchen, wie und mit wieviel Aufwand der dynamische Energieverbrauch optimiert werden könnte. Dieser machte im Experiment zwar mit 22% einen kleineren Anteil aus (vgl. Abbildung 4.5), aber auch dort sind sicherlich noch Optimierungen möglich.

Eine weitere mögliche Forschungsrichtung wäre das Erhöhen der maximalen Anzahl der Pods, welche pro Nodes gescheduled werden können. Gemäss der offiziellen Dokumentation[25] ist diese Limite von Kubernetes momentan auf 110 Pods gesetzt. In der Kubernetes-Distribution Red Hat ist es möglich, die Limite auf 500 Pods per Node zu erhöhen[26]. Momentan wird von Red Hat untersucht, ob diese Limite auf 2500 Pods per Node erhöht werden kann[27]. Dies ist insofern spannend, da aus Abbildung 4.9 hervorgeht, dass es auf dem Test-Cluster sehr viele kleine Pods gibt. Somit ist es möglich, dass sowohl ein Node noch nicht ansatzweise ausgelastet ist und trotzdem nicht weitere Pods ausführen kann, da bereits die Limite von 500 Pods auf diesem Node erreicht wurde.

Abbildungsverzeichnis

Quellenverzeichnis

[1] A. Pols und P. Heidkamp, „Cloud-Monitor 2020, Eine Studie von Bitkom Research im Auftrag von KPMG“, 23. Juni 2020, Bitkom Research GmbH und KPMG AG.

[2] B. Rohleder, „Cloud-Report 2023, Welche Rolle spielt die Cloud für die deutsche Wirtschaft“, 16. Mai 2023, Bitkom Research GmbH.

[3] P. Townend, S. Clement, D. Burdett u. a., „Improving Data Center Efficiency Through Holistic Scheduling In Kubernetes“, S. 1–12, 4. Apr. 2019, Publisher: IEEE. Adresse: https://ieeexplore.ieee.org/document/8705815.

[4] „SIX im Überblick“. (), Adresse: https://www.six-group.com/de/company.html (besucht am 16. März 2024).

[5] „SIX verpflichtet sich zu Net Zero bis spätestens 2050“. (21. Sep. 2022), Adresse: https://www.six-group.com/de/newsroom/media-releases/2022/20220921-six-net-zero.html (besucht am 16. März 2024).

[6] „Offizielles Github von Kubernetes“. (26. Feb. 2024), Adresse: https://github.com/kubernetes/kubernetes/ (besucht am 26. Feb. 2024).

[7] C. Caldato, „Cloud Native for the Enterprise“, 2020, https://www.oracle.com/a/ocom/docs/dc/em/lpd400034389-ebook-artwork-updated.pdf, zuletzt geprüft am 26.02.2024. (besucht am 26. Feb. 2024).

[8] „Offizielle Kubernetes-Dokumentation über Kubernetes Components“. (30. Jan. 2014), Adresse: https://kubernetes.io/docs/concepts/overview/components/ (besucht am 26. Feb. 2024).

[9] „Offizielle Kubernetes-Dokumentation über Kubernetes Workloads“. (12. Juli 2023), Adresse: https://kubernetes.io/docs/concepts/overview/workloads/ (besucht am 26. Feb. 2024).

[10] „Offizielle Kubernetes-Dokumentation über Kubernetes Scheduling“. (14. Dez. 2023), Adresse: https://kubernetes.io/docs/concepts/scheduling-eviction/kube-scheduler/ (besucht am 26. Feb. 2024).

[11] „Offizielle Kubernetes-Dokumentation über Ressourcen Management“. (21. Dez. 2023), Adresse: https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/ (besucht am 16. März 2024).

[12] „Offizielle Kubernetes-Dokumentation über das Operator-Pattern“. (23. Okt. 2023), Adresse: https://kubernetes.io/docs/concepts/extend-kubernetes/operator/ (besucht am 16. März 2024).

[13] „Offizielle Kubernetes-Dokumentation über das Konzept von Controllers“. (22. Nov. 2023), Adresse: https://kubernetes.io/docs/concepts/architecture/controller/ (besucht am 16. März 2024).

[14] „Red Hat-Seite über Helm Charts“. (31. Jan. 2024), Adresse: https://www.redhat.com/en/topics/devops/what-is-helm (besucht am 16. März 2024).

[15] „Kepler Components“. (), Adresse: https://sustainable-computing.io/design/architecture/ (besucht am 26. Feb. 2024).

[16] „Kepler Deep-Dive“. (), Adresse: https://sustainable-computing.io/usage/deep_dive/ (besucht am 26. Feb. 2024).

[17] „Kepler Metrics Overview“. (), Adresse: https://sustainable-computing.io/design/metrics/ (besucht am 26. Feb. 2024).

[18] „Kepler Installation Strategies“. (), Adresse: https://sustainable-computing.io/installation/strategy/ (besucht am 26. Feb. 2024).

[19] „Prometheus Exporter Quickstarts“. (), Adresse: https://grafana.com/oss/prometheus/exporters/ (besucht am 21. März 2024).

[20] „Offizielle Dokumentation über Prometheus“. (), Adresse: https://prometheus.io/docs/introduction/overview/ (besucht am 21. März 2024).

[21] A. James und D. Schien, „A Low Carbon Kubernetes Scheduler“, Jg. 2382, S. 1–10, 6. Sep. 2019, University of Bristol, UK. Adresse: https://ceur-ws.org/Vol-2382/ICT4S2019_paper_28.pdf (besucht am 24. Feb. 2024).

[22] „Github des Vertical Pod Autoscalers von Red Hat/OpenShift“. (), Adresse: https://github.com/openshift/vertical-pod-autoscaler-operator (besucht am 11. März 2024).

[23] „Offizielle Dokumentation von Red Hat zur Installation des VPA in OpenShift“. (), Adresse: https://docs.openshift.com/container-platform/4.12/nodes/pods/nodes-pods-vertical-autoscaler.html#nodes-pods-vertical-autoscaler-install_nodes-pods-vertical-autoscaler (besucht am 11. März 2024).

[24] „Quickstart-Section im Github des Vertical Pod Autoscalers von Kubernetes“. (), Adresse: https://github.com/kubernetes/autoscaler/tree/master/vertical-pod-autoscaler#quick-start (besucht am 11. März 2024).

[25] „Offizielle Kubernetes Dokumentation bezüglich grossen Clusters“. (12. Jan. 2023), Adresse: https://kubernetes.io/docs/setup/best-practices/cluster-large/ (besucht am 21. März 2024).

[26] „OpenShift Scale: Running 500 Pods Per Node“. (10. März 2020), Adresse: https://www.redhat.com/en/blog/500_pods_per_node (besucht am 21. März 2024).

[27] „Running 2500 pods per node on OCP 4.13“. (22. Aug. 2023), Adresse: https://www.redhat.com/en/blog/running-2500-pods-per-node-on-ocp-4.13 (besucht am 21. März 2024).