Analyse der Auswirkungen verschiedener Anwendungsarchitekturen auf den CO_2-Fußabdruck von Cloud-Anwendungen am Beispiel von Microsoft Azure

Bachelorarbeit

für die Prüfung zum

Bachelor of Science

des Studiengangs Informatik

Studienrichtung Informatik

an der

Dualen Hochschule Baden-Württemberg Karlsruhe

von

David Retzlaff

18. September 2022

Ausbildungsfirma: bluehands GmbH & Co.mmunication KG, Karlsruhe

Betreuer der Ausbildungsfirma: Dipl.-Phys. Aydin Mir Mohammadi

Gutachter der Studienakademie: Dr. rer. nat. Mathias Landhäußer

Eigenständigkeitserklärung

(gemäß §5(3) der „Studien- und Prüfungsordnung DHBW Technik“ vom 29. 9. 2015) Ich versichere hiermit, dass ich meine Bachelorarbeitmit dem Titel: „Analyse der Auswirkungen verschiedener Anwendungsarchitekturen auf den CO_2-Fußabdruck von Cloud-Anwendungen am Beispiel von Microsoft Azure“ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Karlsruhe, 18. September 2022

Ort, Datum Unterschrift: David Retzlaff

Abstract

In dieser Arbeit wird untersucht, wie die verschiedenen Bausteine einer Anwendungsarchitektur dazu verwendet werden können, um den CO_2-äquivalenten Fußabdruck einer Anwendung, die in der Microsoft Azure-Cloud gehostet wird, zu minimieren. Darauf basierend wird eine Handlungsempfehlung zu erstellt, die entwickelnden Personen bei der Wahl der Bausteine einer Anwendungsarchitektur helfen soll, die gesamten CO_2-äquivalenten Emissionen zu reduzieren. Durch die Permutation der Bausteine werden unterschiedliche Szenarien anhand einer exemplarischen Web-Shop-Anwendung definiert, die in einer Simulation, welche unterschiedliche Auslastungsszenarien berücksichtigt, untersucht werden. Durch Application Insights werden die Messergebnisse erfasst und durch eine angepasste Methodik nach Cloud Carbon Footprint in CO_2-äquivalente Emissionen umgerechnet. Die berechneten CO_2-äquivalente Emissionen werden den abgeschätzten Kosten, der gemessenen Performance und Usability gegenübergestellt. Dabei stellt sich heraus, dass das primäre Ziel zum Zweck der Optimierung der CO_2-äquivalenten Emissionen, die möglichst effiziente Nutzung der provisionierten Cloud-Ressourcen ist. Unter Berücksichtigung der getroffenen Annahmen zeigt sich vor allem, dass von der Verwendung der Azure Functions abgesehen werden sollte, da diese zu viele Cloud-Ressourcen belegen.

This work investigates, how the different building blocks of an application architecture can be used to reduce the The carbon equivalent footprint of an application, hosted in the Microsoft Azure cloud. Based on the analysis a recommendation for action is created to help developers choose between the available building blocks in order to reduce the carbon emissions. By permuting the building blocks, different scenarios are defined based on an exemplary web store application. These scenarios also take into account different workload scenarios. Through Application Insights, the measurement results are recorded and converted into carbon equivalent emissions using the adapted methodology of the Cloud Carbon Footprint software. The calculated carbon equivalent emissions are compared to the estimated costs, the measured performance and usability. It turns out that the primary goal for the purpose of carbon equivalent emissions is to use the provisioned cloud resources as efficiently as possible. efficient use of provisioned cloud resources. Taking into account the assumptions made, it becomes apparent above all, that the Azure Functions should not be used, because they occupy too many cloud resources.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ID	Identifier
SQL	Structured Query Language
API	Application Programming Interface
CPU	Central Processing Unit
vCPU	Virtual Central Processing Unit
CQRS	Command Query Responsibility Segregation
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IaaS	Infrastructure as a Service
SaaS	Software as a Service
PaaS	Platform as a Service
VM	virtuelle Maschine
CDN	Content Delivery Network
ACU	Azure-Compute-Einheit
NUMA	Non-Uniform Memory Access
SSD	Solid-State-Drive
HDD	Hard-Disk-Drive
TDP	Thermal-Design-Power
SPEC	Standard Performance Evaluation Corporation
SDK	Software Development Kit
CSV	Comma-separated Values
PUE	Power Usage Effectiveness

1 Forschungsfrage

Diese Bachelorarbeit untersucht den CO_2-äquivalenten Fußabdruck von beispielhaften Standard-Cloud-Anwendungen unter Berücksichtigung möglicher Anwendungsarchitekturen. Als Standardanwendungsfall werden die verschiedenen Szenarien eines Web-Shops verwendet. Die Basisarchitektur der zu untersuchenden Web-Shop-Anwendung wird auf eine Microservice-Architektur mit Command Query Responsibility Segregation (CQRS) festgelegt. Die Bausteine der Anwendungsarchitektur, die im Rahmen der Untersuchung betrachtet werden, sind die in der Microsoft Azure-Cloud verfügbaren Hosting-Umgebungen sowie unterschiedliche Kommunikationsmuster bzw. Kommunikationstechnologien zwischen den einzelnen Services.

Das primäre Ziel der Untersuchung ist es, herauszufinden, wie die verschiedenen Bausteine der Anwendungsarchitektur dazu verwendet werden können, um den CO_2-äquivalenten Fußabdruck einer Anwendung, die in der Microsoft Azure-Cloud gehostet wird, zu minimieren und darauf basierend eine Handlungsempfehlung zu erstellen.

Ein weiterer Teilaspekt der Untersuchung ist die Bewertung der Performance, Benutzbarkeit (Usability) und Kosten der Shop-Anwendung gegenüber den CO_2-äquivalenten Emissionen. Daraus resultieren die folgenden Teilfragen:

- Welchen Einfluss hat die Wahl der Hosting-Umgebung auf die CO_2-Emissionen, Performance und Benutzbarkeit?
- Welchen Einfluss hat die Wahl des Kommunikationsmusters bzw. Kommunikationstechnologie auf die CO_2-Emissionen, Performance und Benutzbarkeit?
- Wie stehen die Kosten der Hosting-Umgebungen in Relation zu den CO_2-Emissionen?

Bei der Bewertung wird versucht, nicht nur die direkten CO_2-Emissionen durch den Energieverbrauch zu betrachten, sondern den gesamten Ressourcenverbrauch der Anwendung in Form von CO_2-äquivalenten Emissionen miteinzubeziehen.

2 Aktueller Stand

Aus einem Beitrag der Google Cloud Dokumentation geht hervor, dass die Wahl der Cloud-Regionen die effektivste und einfachste Möglichkeit ist, um die CO_2-Emissionen zu reduzieren. Es muss lediglich eine Cloud-Region mit einer niedrigen Kohlenstoffintensität des lokalen Stromnetzes gewählt werden. Außerdem wird erläutert, dass serverlose Hosting-Umgebungen durch Funktionen, wie Autoscaling und Größenanpassung, weniger inaktive Ressourcen verursachen, als es bei VMs der Fall ist. Ein Beispiel dafür ist die serverlose Hosting-Umgebung Cloud Run, in der containerisierte Anwendungen schneller und mit weniger inaktiven Ressourcen skaliert werden können. Durch weniger inaktive Ressourcen werden entsprechend weniger CO_2-Emissionen verursacht und zusätzlich können Kosten gespart werden. Des Weiteren wird empfohlen monolithische Anwendungen in eine Microservice-Architektur zu überführen, da diese die zuvor genannten Funktionen (Autoscaling und Größenanpassung) besser und schneller nutzen kann. [1]

Insgesamt ist die Erfassung und Bewertung der CO_2-äquivalenten Emissionen jedoch ein sehr neues Gebiet und es entwickelt sich erst in den letzten Jahren langsam die Aufmerksamkeit für die Thematik. Das sogenannte Greenhouse Gas Protocol wurde zwar schon im Jahre 2004 veröffentlicht, ein Werkzeug für das Monitoring der CO_2-äquivalenten Emissionen auf der Basis des GHG Protocol wird von Microsoft für die Azure Cloud jedoch erst im Jahre 2020 veröffentlicht [2]. Eine Untersuchung des GHG Protocol in Bezug auf die Umsetzbarkeit bzw. Umsetzung der Cloud-Provider zeigt jedoch, dass es insbesondere bei Microsoft Azure sehr große Transparenzprobleme gibt [3]. Dies erschwert die Bewertung der CO_2-äquivalenten Emissionen, welche durch die Nutzung von Cloud-Dienstleistung verursacht werden. Auch eine standardisierte Methodik für die Berechnung Software-CO_2-Intensität existiert bislang nicht. Erst im Jahr 2021 hat die Green Software Foundation einen Entwurf für die Bewertung, die Software Carbon Intensity (SCI), welche sich derzeit (2022) immer noch in einer Alpha-Version befindet [4]. Die Motivation der Green Software Foundation ist es damit die bestehende Lücke eines Industriestandards zu füllen.

3 Erfassen von Treibhausgasemissionen

Ein zentraler Aspekt der Untersuchung ist die Erfassung und Auswertung von CO_2-äquivalenten Emissionen. Diese werden als Bewertungsgrundlage benötigt. Im Folgenden werden einige Methodiken und Werkzeuge für die Erfassung der CO_2-äquivalenten Emissionen aufgezeigt.

3.1 Scopes nach GHG Protocol

Mit dem Ziel bessere operative Grenzen festzulegen und um Vergleichbarkeit herstellen zu können, definiert der Standard des GHG Protocol unterschiedliche Emissionsbereiche (Scopes). Diese Scopes ermöglichen die Abgrenzbarkeit zwischen direkten und indirekten Emissionen, die durch ein Unternehmen verursacht werden. Es wird zwischen drei Scopes unterschieden: Scope 1, Scope 2 und Scope 3. [5, p. 25]

Scope 1-Emissionen sind definiert als Treibhausgasemissionen, die direkt durch ein Unternehmen verursacht und dementsprechend auch kontrolliert werden. Dazu zählen bspw. Treibhausgasemissionen durch Fahrzeuge, (Gas-) Heizungen, Öfen oder chemische Prozesse. [5, p. 25]

Scope 2-Emissionen sind definiert als Treibhausgasemissionen, die indirekt durch die Verwendung von Elektrizität durch ein Unternehmen verursacht werden. Der Ausstoß der Treibhausgasemissionen erfolgt bei dem Generierungsprozess der Elektrizität. [5, p. 25]

Scope 3-Emissionen sind definiert als Treibhausgasemissionen aus allen sonstigen indirekten Quellen, die ein Unternehmen dementsprechend indirekt verursacht. Diese Treibhausgasemissionen stammen aus Quellen, die nicht direkt durch das Unternehmen kontrolliert werden können. Dazu gehören bspw. Treibhausgasemissionen, die durch den Transport von Gütern, die Verwendung von Gütern oder Dienstleistung verursacht werden. [5, p. 25]

3.2 Werkzeuge zur Emissionserfassung

Die gängigsten Cloud-Anbieter, also konkret Microsoft Azure, Amazon AWS, und Google Cloud bieten bereits einfache Werkzeuge für die Messung des CO_2-äquivalenten Fußabdrucks an [2, 6, 7].

Im Falle von Microsoft Azure gibt es das Emissions Impact Dashboard, welches über ein Power BI Dashboard als Anwendung bereitgestellt wird [2]. Aufgrund dessen, dass dieses Dashboard nur von Microsoft Enterprise Kunden verwendet werden kann, ist es leider nicht möglich dieses Werkzeug im Rahmen der Untersuchungen zu verwenden [8].

Ein alternativer Ansatz für die CO_2-Emissionserfassung ist die Cloud Carbon Footprint-Software. Dabei handelt es sich um ein Open-Source-Projekt, welches öffentlich unter der Lizenz Apache License 2.0 auf GitHub verfügbar ist. [9, 10]

3.3 Methodik nach Cloud Carbon Footprint

3.3.1 Einführung in die Methodik

Grundsätzlich setzen sich die CO_2-Emissionen, die von einem Cloud-Rechenzentrum verursacht werden aus der Summe der Scope 1, 2 und 3 CO_2-Emissionen (siehe 3.1.), die im Rahmen des Betriebes anfallen, zusammen. Diese Summe bildet in Abhängigkeit von der Verwendung der Rechenzentrumskapazität wiederum die Scope 3 CO_2-Emissionen eines Unternehmens, welche die entsprechenden Cloud-Dienstleistungen in Anspruch nimmt.

Die Cloud Carbon Footprint-Software versucht die CO_2-Emissionen über den Verbrauch einzelner Cloud-Ressourcen zu ermitteln. Dafür werden die Rechnungs- bzw. Verbrauchsdaten über eine API abgerufen und mit vorberechneten Koeffizienten verrechnet. Das Ergebnis der Berechnung ist eine ganzheitliche Abschätzung der durch die Verwendung verursachten CO_2-äquivalenten Emissionen. [11]

3.3.2 Verbrauchsdaten der Azure Cloud

Im Falle von Microsoft Azure verwendet die Cloud Carbon Footprint-Software die Abrechnungs- bzw. Verwendungsdaten, welche über die Azure Consumption APIs bereitgestellt werden. [11]

Die Azure Consumption APIs stellt Verwendungsdaten pro Abonnement als Liste aggregierter Daten über einen Zeitraum von einem Tag bereit. Jeder Eintrag einer solchen Liste beschreibt die Verbrauchsdaten einer Verbrauchskategorie von einer Anwendungsinstanz bzw. Ressourcen-

gruppe (bspw. ein Azure App Service). Dies bedeutet, dass zu einer Instanz potenziell mehrere Listeneinträge pro Tag auftreten können, abhängig von der verwendeten Dienstleistung.

Ein beispielhafter Listeneintrag (siehe Abb. 1, Seite 6) beinhaltet immer den Beginn und das Ende der Verwendung (Zeile 35 und 37), einer zugehörigen Abonnement-ID (Zeile 31) und Informationen zur Identifikation der zugehörigen Instanz (Zeile 11 und 13). Für die Cloud Carbon Footprint-Software sind aber besonders die Details (Zeile 16-26) von Interesse. Dort steht in welcher Region der Service gehostet wird, um welche Art von Service es sich handelt (Zeile 17), welche Hardwarekategorie (Zeile 19-20 und 23) für das Hosting des Services verwendet wird und in welcher Abrechnungseinheit (Zeile 25) die Verbrauchsdaten gegeben sind. Die tatsächlich verbrauchte Menge der Ressource wird als Quantität (Zeile 36) in Abhängigkeit der gegebenen Abrechnungseinheit angegeben. In diesem Fall beträgt der Verbrauch gerundet 0,031 Stunden also ca. 1,86 Minuten über den Zeitraum von 24 Stunden. Da es sich um einen Azure App Service, also eine Web-Anwendung handelt, ist davon auszugehen, dass es sich hierbei um die von der Anwendung verbrauchte vCPU-Zeit handeln muss.

Nach der Methodik der Cloud Carbon Footprint-Software, welche auf der Methodik von Etsy für die Berechnung des CO_2-äquivalenten Fußabdrucks aufbaut, werden die Verwendungsdaten in vier Verwendungskategorien eingeteilt: Rechenzeit, Datenspeicherung, Netzwerk und Arbeitsspeicher [12]. Diese Kategorien werden die Cloud-Juwelen genannt [12]. Die Einteilung in diese Kategorien erfolgt über die entsprechende Abrechnungseinheit. So wird angenommen, dass Rechenzeit in Stunden, Datenspeicherung in Datenmenge pro Zeitspanne, Netzwerk in Datenmenge und Arbeitsspeicher in GB pro Sekunden bzw. Stunden angegeben wird [13]. Alle sonstigen Abrechnungseinheiten werden nicht beachtet. Innerhalb jeder dieser Kategorien wird die Berechnung der CO_2-Emissionen gesondert durchgeführt und die Ergebnisse im Anschluss aggregiert. [11]

Die Berechnung der CO_2-äquivalenten Emissionen wird im Folgenden näher erläutert.

3.3.3 Aufstellung der Koeffizienten für die Energieberechnung

Ein essenzieller Teil der Methodik ist die Umrechnung der Verbrauchsdaten in Energie. Die Aufstellung der für die Umrechnung benötigen Koeffizienten wird im Folgenden beschrieben.

```
{
    "accountName": null,
    "additionalProperties": null,
    "billableQuantity": "None",
    "billingPeriodId": "/subscriptions/0f4aef05-7409-4267-a0d8-
    ecb192742007/providers/Microsoft.Billing/billingPeriods/202207",
    "consumedService": "microsoft.web",
    "costCenter": null,
    "currency": "EUR",
    "departmentName": null,
    "id": "/subscriptions/0f4aef05-7409-4267-a0d8-ecb192742007/providers/Microsoft
    .Billing/billingPeriods/202207/providers/Microsoft
    .Consumption/usageDetails/a6900260-636c-deb1-a227-f6c9895d1af0",
    "instanceId": "/subscriptions/0f4aef05-7409-4267-a0d8-
    ecb192742007/resourcegroups/azure-functions-scaling-behaviour-
    test_group/providers/microsoft.web/sites/azure-functions-scaling-behaviour-test",
    "instanceLocation": "EU West",
    "instanceName": "azure-functions-scaling-behaviour-test",
    "invoiceId": null,
    "isEstimated": true,
    "meterDetails": {
      "meterCategory": "Azure App Service",
      "meterLocation": "EU West",
      "meterName": "F1",
      "meterSubCategory": "Free Plan - Linux",
      "pretaxStandardRate": "None",
      "serviceName": "Azure App Service",
      "serviceTier": "Azure App Service Free Plan - Linux",
      "totalIncludedQuantity": "None",
      "unit": "1 Hour"
    },
    "meterId": "a90aec9f-eecb-42c7-8421-9b96716996dc",
    "name": "a6900260-636c-deb1-a227-f6c9895d1af0",
    "pretaxCost": "0",
    "product": "",
    "subscriptionGuid": "0F4AEF05-7409-4267-A0D8-ECB192742007",

    "subscriptionName": "Visual Studio Enterprise - MPN",
    "tags": null,
    "type": "Microsoft.Consumption/usageDetails",
    "usageEnd": "2022-07-04T23:59:59Z",
    "usageQuantity": "0.032255999999999999949817919286942924372851848602294921875",
    "usageStart": "2022-07-04T00:00:00Z"
}
```


Abbildung 1: Azure Consumption Usage Details - List

Je nach Art der Hardware, auf der die Software ausgeführt wird, kann der Stromverbrauch stark variieren. Um eine Aussage über den Stromverbrauch von Software treffen zu können, muss dementsprechend bekannt sein, auf welcher Hardware sie ausgeführt wird. In einem On-premise Rechenzentrum ist dies sehr einfach, da all diese Daten einsehbar sind. In einer Cloud-Umgebung ist dies üblicherweise nicht der Fall, da die Cloud-Anbieter versuchen konkrete Hardware zu abstrahieren und stattdessen Rechenleistung als Dienstleistung verkaufen. Microsoft macht dies in der Azure Cloud über die sogenannte Azure-Compute-Einheit [14]. Das Ziel der Azure-Compute-Einheit (ACU) ist es, eine Vergleichbarkeit der Rechenleistung unterschiedlicher Prozessoren herzustellen [14]. Der Nachteil dieser Abstraktion ist jedoch, dass Microsoft bei vielen Dienstleistungen lediglich die ACU angibt. Dies ist bspw. bei Azure App Service Plänen der Fall [15]. Dadurch ist die zugrundeliegende Hardware nicht einsehbar.

Bei virtuellen Maschinen variieren die verfügbaren Daten ebenfalls stark. In manchen Fällen ist der Prozessor eindeutig bekannt bspw. werden virtuelle Maschinen (VMs) der Dv5- und Dsv5-Serie auf der dritten Generation Intel® Xeon® Platinum 8370C (Ice Lake) Prozessoren ausgeführt [16]. Die tatsächliche Hardwarekonfiguration eines Servers ist jedoch nicht bekannt. Die maximal verfügbaren VM-Größen lassen zwar teilweise darauf schließen, wie viele Non-Uniform Memory Access (NUMA)-Knoten und Arbeitsspeicher in einem Server mindestens vorhanden sind, die tatsächliche Anzahl kann davon jedoch abweichen. Da die Central Processing Unit (CPU) Intel® Xeon® Platinum 8370C nicht in der offiziellen Liste der Ice Lake Serie zu finden ist, höchstwahrscheinlich, weil es sich bei der CPU um ein OEM-Produkt handelt, ist selbst eine Abschätzung über die Mindestanzahl der NUMA-Knoten nicht eindeutig möglich [17]. Über den Wirkungsgrad des Servernetzteils oder sonstige Hardware eines Servers lässt sich selbstverständlich überhaupt keine Aussage treffen. Dasselbe gilt für den persistenten Speicher eines Servers, da dieser hardwaretechnisch sehr stark variieren kann. Eine Speichererweiterung ist bspw. durch einen simplen Austausch der aktuellen SSDs bzw. HDDs möglich.

Für den Fall, dass doch bekannt ist, auf welcher Hardware die Software ausgeführt wird, gibt es eine weitere Problematik. Die CPU-Hersteller AMD und Intel geben bei ihren Prozessoren keine öffentlich verfügbaren eindeutigen Angaben zu dem Stromverbrauch an. Lediglich die Thermal-Design-Power (TDP) wird angegeben. Diese ist jedoch nur ein Richtwert für die thermische Verlustleistung eines Prozessors unter bestimmten Bedingungen. Die tatsächliche Energieaufnahme

einer CPU verhält sich in der Realität häufig anders und ist von unterschiedlichen Faktoren abhängig. Ein maßgeblicher Faktor ist die Auslastung. In einem On-premise Rechenzentrum wäre die Energieaufnahme leicht zu ermitteln, indem bspw. ein Leistungsmessgerät zwischen die Hardware geschaltet wird. Allerdings ist es selbst dann schwierig den Energieverbrauch einer einzelnen CPU zu isolieren. Auch die Betreibenden eines Cloud-Rechenzentrums sind selbstverständlich daran interessiert, wie viel Energie von der Hardware verbraucht wird und es ist davon auszugehen, dass ihnen dies auch bekannt ist. Im Detail verfügbar sind diese Daten im Falle von Microsoft Azure jedoch nicht. Microsoft veröffentlicht lediglich eine jährliche Übersicht über den gesamten Energieverbrauch des Unternehmens [18].

Transparenz bedingt ist demnach eine Methodik erforderlich, die trotz der schwachen Datenlage eine Abschätzung der CO_2-äquivalenten Emissionen ermöglicht.

Ein Industriestandard für die Bestimmung der Energieaufnahme eines Servers ist der SPECpower_ssj2008 Benchmark entwickelt von Non-Profit Organisation Standard Performance Evaluation Corporation (SPEC). Im Rahmen des Benchmarks wird zunächst der maximale Gesamtdurchsatz eines Systems ermittelt. Anschließend wird in 10%-Schritten des maximalen Gesamtdurchsatzes konstant der Durchsatz des Systems erhöht und dabei die Energieaufnahme durch ein externes Messgerät gemessen. Dies hat zur Folge, dass die gesamte Energieaufnahme des Systems ermittelt wird inklusive der Verlustleistung von Netzgeräten, Arbeitsspeicher und Festplatten bzw. SSD-Speicher. Da das Ziel des Benchmarks nicht die Messung der Energieaufnahme von persistentem Speicher ist, sind die Testsysteme mit einem Minimum an Speicher ausgestattet. Alle Testsysteme werden durch eine Luftkühlung gekühlt. [19]

Für die Umrechnung der Rechenzeit in die Energieaufnahme eines Systems wird der Stromverbrauch im inaktiven Zustand und der Stromverbrauch unter Volllast benötigt.

Die Methodik der Cloud Carbon Footprint-Software zur Berechnung der Energiekoeffizienten gruppiert zunächst alle Serversysteme der SPECpower_ssj2008 Benchmarks nach Prozessormikroarchitektur. In dem Beispiel wird nach der AMD EPYC Gen 1 Mikroarchitektur gruppiert (siehe Abb. 2, Seite 9). Anschließend wird für jede Konfiguration innerhalb der Gruppe der minimale Stromverbrauch und der maximale Stromverbrauch pro Thread sowie der verfügbare Arbeitsspeicher pro Chip bestimmt. Zuletzt wird der Mittelwert für den Stromverbrauch im inaktiven

und voll ausgelasteten Zustand sowie Arbeitsspeicher pro Chip über die gesamte Gruppe der Prozessormikroarchitektur bestimmt. [20]

```
amd_epyc_gen1 = {}
amd_epyc_gen1['Idle watts'] = (servers_amd_epyc_gen1['avg. watts @ active
↪  idle'].astype(float) / servers_amd_epyc_gen1['Total Threads']).mean()
amd_epyc_gen1['100\% watts'] = (servers_amd_epyc_gen1['avg. watts @
↪  100\%'].astype(float) / servers_amd_epyc_gen1['Total Threads']).mean()
amd_epyc_gen1['GB/Chip'] = (servers_amd_epyc_gen1['Total Memory (GB)'] /
↪  servers_amd_epyc_gen1['Chips']).mean()
amd_epyc_gen1['Total memory'] = servers_amd_epyc_gen1['Total Memory (GB)']

```

Quelle: [20]

Abbildung 2: Berechnung der Energiekoeffizienten für die AMD EPYC Gen 1 Mikroarchitektur

Es ist davon auszugehen, dass die Gruppierung in Prozessormikroarchitekturen durchgeführt wird, da zum einen nicht für alle auf dem Markt verfügbaren CPU-Modelle Benchmarks vorhanden sind und zum anderen bspw. im Falle von Microsoft Azure die genannten Probleme der Transparenz über die verwendete Hardware bestehen. Bei virtuellen Maschinen der Microsoft Azure Cloud wird häufig nur eine Gruppe von möglicher Hardware, auf der die VMs gehostet werden, angegeben.

Der online verfügbare Datensatz der SPECpower_ssj2008 Benchmarks ist mit derzeitig ca. 726 (Stand 2022) zudem nicht besonders groß. Aufgeführt werden ca. 250 unterschiedliche Prozessormodelle in Serversystemen unterschiedlicher Hersteller bzw. Konfigurationen.

Die Folge davon ist, dass lediglich eine grobe Abschätzung des realen Energieverbrauchs möglich ist [12]. Zudem ist es nahezu unmöglich zu sagen, wie stark die Berechnung mithilfe der ermittelten Energiekoeffizienten von den realen Werten abweichen [12]. Besonders dann, wenn es keine verfügbaren Benchmarks für ein spezifisches CPU-Modell gibt oder wenn lediglich eine Gruppe möglicher für das Hosting verwendeter CPU-Modelle bekannt ist, wie es bspw. bei manchen virtuellen Maschinen in der Microsoft Azure Cloud der Fall ist.

Ein weiteres Problem des SPECpower_ssj2008 Benchmarks ist, dass die Energiemessung den gesamten Stromverbrauch eines Serversystems ermittelt [19]. Damit ist die Isolierung des Stromverbrauchs einzelner Hardwarekomponenten nahezu unmöglich. Die ermittelte durchschnittliche

Stromaufnahme pro Thread beinhaltet demnach auch insbesondere die Stromaufnahme des Arbeitsspeichers.

Dennoch ist die genannte Methodik die derzeitig Beste, die vollständig öffentlich zugänglich ist und zudem die Einzige, deren Methodik vollständig einsehbar ist.

3.3.4 Umrechnung der Rechenzeit in Energie

In dem ersten Schritt der Methodik wird der durchschnittliche Stromverbrauch in Watt $\overline{W}$ der verwendeten Prozessormikroarchitektur pro vCPU abgeschätzt (1). Für den Fall, dass die Prozessormikroarchitektur nicht eindeutig bekannt ist, verwendet die Cloud Carbon Footprint-Software den Durchschnitt einer Gruppe aus Prozessormikroarchitekturen, sofern diese bekannt ist. Für den Fall, dass falls selbst die nicht bekannt ist, wird der Median aller Prozessormikroarchitekturen als Ersatzwert verwendet, mit dem Ziel keine Überabschätzung zu machen. Da Server auch im Leerlauf Energie benötigen wird in der Berechnung der minimale Stromverbrauch $W_{\text{min.}}$ mit dem durch zusätzliche Auslastung verursachten Stromverbrauch addiert. Der durch die Auslastung verursachte Stromverbrauch ergibt sich durch die Multiplikation der durchschnittlichen Virtual Central Processing Unit (vCPU) Auslastung $\overline{\text{vCPU}\%}$ über die gemessene Zeitspanne mit dem mittleren Stromverbrauch der CPU, welche über die Differenz des maximalen Stromverbrauchs $W_{\text{max.}}$ und des minimalen Stromverbrauchs $W_{\text{min.}}$ berechnet wird. [11]

$$\overline{W} = W_{\text{min.}} + \overline{\text{vCPU}\%} \cdot (W_{\text{max.}} - W_{\text{min.}}) \tag{1}$$

Die Formel (1) geht von einer linearen Zunahme der Leistung in Abhängigkeit der durchschnittlichen vCPU-Auslastung aus.

In dem zweiten Schritt der Methodik wird der durchschnittliche Stromverbrauch in Watt $\overline{W}$ mit der tatsächlichen vCPUs-Zeit in Stunden h multipliziert (2). Dies ergibt eine Abschätzung des Energieverbrauchs Wh in Abhängigkeit von der Dauer der Verwendung h in Wattstunden. [11]

$$Wh = \overline{W} \cdot h \tag{2}$$

Im Falle von Microsoft Azure wird davon ausgegangen, dass die Verbrauchsdaten nicht die Anzahl der tatsächlichen vCPUs beinhalten. Aus diesem Grund wird der berechnete Wert zusätzlich mit der Anzahl der vCPUs einer Instanz multipliziert, um den gesamten Energieverbrauch abzuschätzen. [11]

3.3.5 Umrechnung der Arbeitsspeicherauslastung in Energie

Da die Cloud Carbon Footprint-Software für die Berechnung des Energieverbrauchs die Koeffizienten des SPECpower_ssj2008 Benchmarks verwendet (siehe 3.3.3.) und diese bereits den Energieverbrauch des Arbeitsspeichers beinhalten, wird eine gesonderte Berechnung des Energieverbrauchs des Arbeitsspeichers nur dann durchgeführt, wenn die provisionierte Ressource deutlich mehr Arbeitsspeicher verwendet, als die Testsysteme des Benchmarks [11].

Für die im Rahmen der Simulation betrachten Szenarien ist dies nicht relevant, da diese weit unterhalb dieser Grenze liegen.

3.3.6 Berechnung der Scope 2-Emissionen

Ein essenzieller Teil der Methodik ist die Umrechnung der verbrauchten Energie in CO_2-äquivalente Scope 2-Emissionen.

Die Cloud Carbon Footprint-Software macht dies, indem die gemessenen Wattstunden Wh mit der Division durch 1000 zunächst in Kilowattstunden (kWh) umgerechnet und dann mit der Power Usage Effectiveness (PUE) und den Energieemissionsfaktoren in Tonnen CO_2-äquivalent tCO_2e/kWh multipliziert werden (3). Das Ergebnis der Berechnung (3) sind die CO_2-äquivalenten Scope 2-Emissionen in Tonnen. [11, 10]

$$tCO_2e = \frac{Wh}{1000} \cdot PUE \cdot tCO_2e/kWh \tag{3}$$

Die PUE gibt an, wie energieeffizient ein Rechenzentrum ist. Der bestmögliche Wert ist dabei 1, was bedeuten würde, dass die verbrauchte Energie direkt für den Betrieb der Hardware verwendet werden kann. In der Realität geht jedoch viel Energie für die Kühlung der Hardware verloren, weshalb dieser Wert meistens leicht oberhalb der 1 liegt. [11]

3.3.7 Berechnung der Scope 3-Emissionen

Die Methodik für die Berechnung der Scope 3-Emissionen der Cloud Carbon Footprint Software basiert auf einer von der Green Software Foundation definierten Methodik [21].

Die Scope 3-Emissionen M werden durch die Multiplikation der gesamten Emissionen der Hardware TE mit dem Verhältnis von der provisionierten Zeit TR zu der gesamten Betriebszeit der Hardware EL und dem Verhältnis zwischen der Anzahl an provisionierter Ressourcen RR zur Gesamtanzahl der verfügbaren Ressourcen TR berechnet (4). [21]

$$M = TE \cdot \frac{TR}{EL} \cdot \frac{RR}{TR} \tag{4}$$

Die besonders schwierige Herausforderung dabei ist, die Bestimmung der gesamten Hardwareemissionen. Die Cloud Carbon Footprint-Software verwendet dafür eine von Teads Engineering entwickelte Methodik zur Abschätzung der gesamten Hardwareemissionen TE. Für die Zeit der Provisionierung TR nutzt die Cloud Carbon Footprint Software die vCPU-Stunden der Verbrauchsdaten. Für die gesamte Lebenszeit der Hardware EL werden einheitlich 4 Jahre verwendet und für die Anzahl an provisionierter Ressourcen RR die Anzahl der provisionierten vCPUs. [21]

4 Isolierte Bausteine einer verteilten Cloud-Anwendung

Ziel der Untersuchung ist die Berechnung und Bewertung des CO_2-äquivalenten Fußabdrucks der einzelnen Bausteine einer Standardanwendungsarchitektur. Diese Grundbausteine sind im Wesentlichen die Hosting-Umgebung, in der die Anwendung ausgeführt wird und das Kommunikationsmuster, welches zur Service-übergreifenden Kommunikation verwendet wird. Diese beiden Bausteine bestimmen maßgeblich, wie die Mikroarchitektur der Anwendung entwickelt werden kann.

Um nicht eine gesamte Web-Shop-Anwendung und deren Geschäftslogik implementieren zu müssen, werden stattdessen einzelne Use-Cases der Anwendung als Szenarien isoliert und durch die

Implementierung simuliert. Ziel der Szenarien ist demnach die Simulation der Services einer verteilten Web-Shop-Anwendung, um die CO_2-äquivalenten Emissionen der Hosting-Umgebungen und Kommunikationsmuster bzw. Kommunikationstechnologien zu untersuchen. Der Focus liegt demnach nicht auf der Simulation einer verteilten Anwendung. Die Szenarien dienen vielmehr der künstlichen Erzeugung einer Auslastung der Ressourcen, welche von der Cloud bereitgestellt werden und der anschließenden Berechnung der dadurch verursachten CO_2-äquivalenten Emissionen.

Als Grundstruktur der fiktiven Shop-Anwendung wird eine Microservice-Architektur mit CQRS gewählt. Neben der funktionalen Dekomposition durch die Microservice-Architektur in einzelne, isolierte und unabhängige Mircoservices, trennt CQRS die einzelnen Mircoservices zusätzlich in Command- und Query-Services.

Ein Query-Service stellt eine Schnittstelle zur Verfügung, welche für das Lesen von Daten zuständig ist und ein Command-Service eine Schnittstelle für das Schreiben von Daten. Die Anwendungsmodelle, welche für das Lesen und Schreiben der Daten zuständig sind, müssen nicht unbedingt von einer Datenquelle bzw. einer Datenbank bedient werden. Dies ermöglicht üblicherweise, die für die einzelnen Services verwendete Technologie sehr granular an den konkreten Anwendungsfall anzupassen und zu optimieren. [22]

4.1 Microsoft Azure Cloud-Hosting

Die Microsoft Azure Cloud bietet unterschiedliche Hosting-Umgebungen an, welche für das Hosting von Mircoservices verwendet werden können. Die drei Abstraktionsstufen des Hostings in der Microsoft Azure Cloud sind Infrastructure as a Service (IaaS), Platform as a Service (PaaS) und Software as a Service (SaaS). SaaS Dienstleistungen sind allerdings für die Bereitstellung eines Web-Shops nicht von Interesse.

Bei dem IaaS-Modell haben die Kunden die meiste Kontrolle über die Cloud-Infrastruktur, müssen sich aber auch selbst um die Konfiguration der bereitgestellten Infrastruktur kümmern. Es wird lediglich die Komplexität für das Verwalten der physischen Hardware umgangen. Zu den Services von IaaS zählen Server in Form von VMs, Speicher, Netzwerke und Firewalls. Es wird somit die Kapazität eines Microsoft Azure Rechenzentrums als Service bereitgestellt. [23]

Die nächste Abstraktionsstufe nach IaaS ist PaaS. In dem PaaS-Modell werden zusätzlich bzw. aufbauend auf den Dienstleistungen von IaaS Dienstleistungen, wie Datenbankmanagement, Middleware, Business Inteligence (BI) und Betriebssysteme in Form von Docker Container-Instanzen bzw. Kubernetes, angeboten. [24]

IaaS reduziert den Konfigurations- und Verwaltungsaufwand im Vergleich zu PaaS deutlich, verringert aber gleichzeitig den Einfluss auf die für die Dienstleistungen verwendete Hardware.

Ein weiteres in der Microsoft Azure Cloud verbreitetes Dienstleistungskonzept ist serverless Computing. serverless Computing eliminiert den Verwaltungsaufwand von Cloud-Anwendungen nahezu vollständig. Cloud-Anwendungen werden automatisch durch den Cloud-Anbieter provisioniert, skaliert und verwaltet. Zu den serverlosen Dienstleistungen der Microsoft Azure Cloud zählen serverlose Funktionen (Azure Functions), serverloses Kubernetes (Azure Kubernetes Service), serverlose Anwendungsumgebungen (Azure App Service), serverlose API-Gateways, serverlose Datenbanken (Azure Cosmos DB und Azure SQL-Datenbank – serverlos), Azure serverless Storage (Azure Blob Storage) und serverloses Messaging (Azure Service Bus). [25, 26]

Die Dienstleistungen bilden die Grundbausteine einer Anwendungsarchitektur von einer Anwendung, die in der Microsoft Azure Cloud gehostet wird.

Für das Hosting der Szenarien einer Shop-Anwendung ergebenen sich aus den genannten Dienstleistungen die Hosting-Umgebungen: VMs, Azure Container Instances, Azure App Services und Azure Functions. Diese sollen in den einzelnen Szenarien isoliert auf ihren CO_2-Fußabdruck untersucht werden.

4.2 Service übergreifende Kommunikation

In Abhängigkeit der verwendeten Hosting-Umgebungen lassen sich unterschiedliche Kommunikationsmuster bzw. Kommunikationsprotokolle für die Service-Kommunikation verwenden.

Im Falle der Shop-Anwendung ist ein zentraler Microservice der Produkt-Query-Service, welcher für das Lesen der Produktdaten zuständig ist. Eine die fiktive Website besuchende Person durchsucht während dem Online-Shopping den Produktkatalog und sendet dabei regelmäßig Anfragen an die Application Programming Interface (API)-Gateway der Anwendung. Die API-Gateway lei-

tet die Leseoperationen an Produkt-Query-Service weiter, welcher diese dann weiter verarbeitet. Bei der Weiterverarbeitung sendet der Produkt-Query-Service Anfragen an die Datenbank. Die aus der Datenbank geladenen Daten werden dann entsprechend der Geschäftsregeln des Services verarbeitet. Dabei werden bspw. der Bruttopreis und ggf. aktuelle Rabatte berechnet. Sobald dies abgeschlossen ist, sendet der Produkt-Query-Service die Ergebnisse zurück an die API-Gateway. Die API-Gateway sendet letztendlich die Ergebnisse zurück an die Client-Anwendung.

Das zugrundeliegende Kommunikationsmuster wird als temporal gekoppelte Kommunikation bezeichnet. Die Client-Anwendung sendet eine Request-Nachricht an die API-Gateway und wartet auf eine Antwort, entweder blockierend oder nicht blockierend, je nach Implementierung. Die API-Gateway sendet ebenfalls eine Nachricht an den Produkt-Query-Service und wartet ebenfalls auf eine Antwort. Der Produkt-Query-Service antwortet der API-Gateway nach Abschluss der Verarbeitung der Nachricht und diese wiederum antwortet letztendlich der Client-Anwendung.

Im Falle der Shop-Anwendung ist das zugrundeliegende Kommunikationsprotokoll zwischen Client-Anwendung und API-Gateway das Hypertext Transfer Protocol (HTTP) bzw. Hypertext Transfer Protocol Secure (HTTPS), welches die Verbindung zwischen den einzelnen Endpunkten so lange offen hält, bis entweder eine Zeitüberschreitung stattfindet oder die Antwort gesendet wird.

Für die dienstübergreifende Kommunikation, auf der Basis der temporal gekoppelte Kommunikation, sind gängige Technologien gRPC und Web-APIs, wozu bspw. GraphQL und REST-APIs gehören. All diese Technologien verwenden ebenfalls das HTTP bzw. HTTPS [27, 28, 29].

Die temporal gekoppelte Kommunikation eignet sich sehr gut für die Bereitstellung eines CQRS Query-Modells, da Client-Anwendungen üblicherweise schnell eine Antwort von dem Server benötigen, wenn sie Daten anfordern. Dies gilt insbesondere für die Client-Anwendung der fiktiven Online-Shop-Anwendung. Ein Problem der temporal gekoppelte Kommunikation ist, dass bei der Verwendung von HTTP die zugrundeliegenden Web-Sockets für die gesamte Dauer einer Anfrage aktiv und verbunden bleiben müssen. Dies hat zur Folge, dass besonders bei vielen gleichzeitigen lang andauernden Anfragen viele Systemressourcen reserviert werden müssen, was dazu führen, kann das ein Server schnell durch viele parallele Anfragen vollständig ausgelastet ist. Dieses Pro-

blem kann nur mittels Lastenverteilung entweder über einen gesonderten Server und durch Client-seitige Lastenverteilung umgangen werden.

Ein alternatives Kommunikationsmuster zur temporal gekoppelte Kommunikation ist die temporal entkoppelte Kommunikation. Diese kann bspw. durch einen unidirektional Nachrichtenaustausch über einen Message-Broker realisiert werden. Die temporal entkoppelte Kommunikation in Kombination mit bspw. dem Competing-Consumers-Pattern eliminiert die Problematik der temporal gekoppelte Kommunikation, indem eine Nachricht an den Message-Broker gesendet wird und dann nicht auf eine unmittelbare Antwort gewartet wird, wodurch keine große Anzahl an Web-Sockets über einen langen Zeitraum gleichzeitig aktiv gehalten werden müssen. Die bei dem Message-Broker eingehenden Nachrichten werden stattdessen zwischengespeichert bis ein Service die Nachricht bearbeitet [30]. Die Anzahl der Nachrichten abarbeitenden Services kann dabei an die Anzahl der eingehenden Nachrichten angepasst werden, wodurch sehr einfach eine sehr effiziente Lastenverteilung ermöglicht wird [31]. Microsoft Azure bietet über den Azure Service Bus solch einen Message-Broker als Dienstleistung an [32]. Die Nachrichten werden in Nachrichtenwarteschlangen (Queues) nach dem FIFO (first in first out) Prinzip zwischengespeichert [32].

Das Competing-Consumers-Pattern ist besonders gut geeignet für die Implementierung eines Command-Services nach CQRS. Dies liegt daran, dass Änderungen an einem Datenbestand häufig mehr Zeit in Anspruch nehmen können und dürfen als bspw. Leseoperationen. In der fiktiven Shop-Anwendung darf beispielsweise die Bearbeitung einer Bestellung über den Bestellungs-Command-Service mehrere Minuten benötigen und es entsteht keine Beeinträchtigung für die im Online-Shop aktive Person. Nach Abschluss der Bearbeitung sendet in diesem Beispiel der Bestellungs-Command-Service eine E-Mail mit dem aktuellen Bestellstatus. Das Einzige, was dabei sichergestellt werden muss, ist, dass die Person, welche die Bestellung abgesetzt hat, über die erfolgreiche Zustellung der Nachricht an den Message-Broker benachrichtigt werden muss, um ihr das Gefühl der direkten Manipulation zu suggerieren.

In der Simulation soll untersucht werden, welchen Einfluss die Kommunikationsmuster bzw. Kommunikationstechnologien auf die CO_2-äquivalenten Emissionen hat.

4.3 Definition der Szenarien mit temporal gekoppelter Kommunikation

Der erste isolierte Use-Case ist der exemplarische Produkt-Query-Service der fiktiven Shop-Anwendung. Um dessen Funktionalitäten weiter einzuschränken, wird in dem Szenario nur die Abfrage einer Produktseite betrachtet. Es wird außerdem festgelegt, dass statische Ressourcen wie bspw. Produktbilder durch Content Delivery Networks (CDNs) bereitgestellt werden. Dementsprechend enthält die Antwort des Produkt-Query-Services auf eine Anfrage nach einer Produktseite lediglich die Produktdaten.

Nach der Methodik der Cloud Carbon Footprint-Software bzw. nach der Methodik, welche von Etsy entwickelt wurde, um den CO_2-Fußabdruck einer Cloud-Anwendung zu ermitteln, fallen drei wesentliche Faktoren ins Gewicht: die Verwendung von CPU-, Arbeitsspeicher- und Netzwerkressourcen [11, 12].

Um die Auslastung der CPU zu simulieren wird für jedes Produkt der Produktseite der Bruttopreis und ein Rabatt berechnet. Um eine möglichst reale Netzwerkauslastung zu simulieren werden die Produktdaten in einer SQL-Datenbank gespeichert und mit jedem Request ohne Caching neu geladen.

Da es sich um einen Query-Service handelt, wird die temporal gekoppelte Kommunikation als Kommunikationsmuster gewählt. Als Implementierungen dieses Kommunikationsmusters wird gRPC und eine Web-API über HTTP in die Untersuchung aufgenommen.

In den Szenarien soll der CO_2-Fußabdruck dieser Technologien in Kombination mit den in der Microsoft Azure-Cloud verfügbaren Hosting-Umgebungen verglichen werden. Diese sind Azure VMs, Azure Container Instances, Azure App Services und Azure Functions. Dabei bleibt die Geschäftslogik und Datenbank in allen Szenarien immer gleich, um eine Vergleichbarkeit der Szenarien zu gewährleisten und um den CO_2-Fußabdruck der Kommunikationstechnologien sowie Hosting-Umgebungen bestmöglich isolieren zu können.

Ziel ist es, über die Szenarien die bestmögliche Kombination aus Hosting-Umgebungen und Kommunikationstechnologie zu finden. Dabei ist die Annahme, dass durch die Wahl der Hosting-

Umgebung und Kommunikationstechnologie letztendlich der gesamte CO_2-äquivalente Fußabdruck einer Anwendung optimiert werden kann.

4.3.1 Szenario 1: Azure App Service und Web-API

In dem Szenario wird der Produkt-Query-Service auf der Basis eines Azure App Services mit einer Web-API entwickelt.

Für den Azure App Service wird ein Linux-basierter App Service Plan mit dem Preismodell P1v3 angelegt. Das Preismodell P1v3 umfasst zwei vCPUs mit einer minimalen ACU von 195, 8 GB Arbeitsspeicher und 250 GB persistenter Speicher, die fest provisioniert sind. Damit ist eine weitere vertikale Skalierung um 2 weitere Stufen möglich. [15]

Der Azure App Service ermöglicht außerdem eine horizontale Skalierung um weitere 30 Instanzen. In dem Szenario wird die Autoskalierung jedoch deaktiviert, um in allen Szenarien möglichst ähnliche Bedingungen zu schaffen.

4.3.2 Szenario 2: Azure App Service und gRPC

Das Szenario wird analog zu dem Szenario 1 (siehe 4.3.1.) konfiguriert, allerdings wird gRPC als Kommunikationstechnologie statt der Web-API für den Produkt-Query-Service verwendet.

4.3.3 Szenario 3: VM und Web-API

In dem Szenario 3 wird wie bei den anderen Szenarien dieselbe Geschäftslogik für den Produkt-Query-Service verwendet.

Die Größe der VM wird an das Szenario 1 angenähert. Dies bedeutet, dass eine VM mit 2 vCPUs und 8 GB Arbeitsspeicher gewählt wird. Konkret wird die Standard_D2ds_v5 VM der Ddsv5-Serie verwendet. Diese läuft auf der dritten Generation der Intel® Xeon® Platinum 8370C Prozessoren. Die Prozessoren basieren auf der Ice Lake Mikroarchitektur. [16]

Aufgrund fehlender Transparenz der Azure App Services ist es nicht möglich die Mikroarchitektur des Azure App Services zu ermitteln. Im Gegensatz dazu, ist bei einigen virtuellen Maschinen,

jedoch nicht bei allen, die Mikroarchitektur bzw. das konkrete CPU-Modell bekannt. Aus diesem Grund ist es nicht möglich aus den verfügbaren Preismodellen der VMs eine zu dem App Service Plan äquivalente Mikroarchitektur zu wählen.

4.3.4 Szenario 4: VM und gRPC

Das Szenario 4 wird analog zu Szenario 3 (siehe 4.3.3.) bezüglich der VM-Konfiguration festgelegt. Statt der Web-API wird jedoch gRPC als Kommunikationstechnologie für den Produkt-Query-Service verwendet.

4.3.5 Szenario 5: Container Instance und Web-API

In dem Szenario 5 wird der Produkt-Query-Service als Azure Container Instance mit Web-API definiert. Als Hosting-Umgebung werden die Azure Container Instanzen verwendet. Die Azure Container Instanz unterstützt 1-4 vCPUs und zwischen 1,5 GB und 16 GB Arbeitsspeicher abhängig von der Region [33]. Um auch in diesem Szenario annähernd ähnliche Bedingungen zu schaffen werden 2 vCPUs und 8 GB Arbeitsspeicher provisioniert.

4.3.6 Szenario 6: Docker-Container und gRPC

Szenario 6 wird bezüglich der Basiskonfiguration analog zu Szenario 5 (siehe 4.3.5.) festgelegt. Statt der Web-API wird jedoch gRPC verwendet.

4.3.7 Szenario 7: Azure Function und Web-API

In Szenario 7 wird der Produkt-Query-Service als eine Azure Function App mit HTTP-Trigger definiert. HTTP-Trigger verhalten sich wie eine Web-API.

Als Hosting-Modell wird der Standard, der Verbrauchsplan gewählt. Dieser umfasst eine automatische Skalierung der einzelnen Hosting-Instanzen abhängig von der aktuellen Auslastung bis hin zu maximal 200 Instanzen, im Falle von Windows-basierten Function-Hosts. Linux-basierte Azure Functions sind auf maximal 100 Hosting-Instanzen limitiert. Die ACU einer Azure Function Hosting-Instanz beträgt 100. Dies entspricht ca. der Hälfte der ACU des in Szenario 1 (siehe

4.3.1.) gewählten App Service Plans. Die Anzahl an vCPUs pro Instanz ist nicht konfigurierbar. Der Arbeitsspeicher einer einzelnen Hosting-Instanz ist auf 1,5 GB limitiert. [34]

Eine Azure Function hat eine standardmäßige Timeoutdauer von 5 Minuten. Die maximale Timeoutdauer kann im Verbrauchsplan auf 10 Minuten gesetzt werden. Dies ist die maximale Laufzeit einer Function bis sie terminiert wird. [34]

Innerhalb des Verbrauchsplans kann es zu einem Kaltstartverhalten kommen, da die Anzahl der Instanzen im Leerlauf möglicherweise auf null herunterskaliert werden. Dies kann zu einer sehr hohen Latenz bei dem Anlaufen der Anwendung führen, da es Zeit benötigt, um neue Hosting-Instanzen bereitzustellen. [34]

Alternative Azure Functions Pläne sind der Premium-Plan, Dezidierter-Plan, App Service Plan, Kubernetes. Die Hosting-Instanzen dieser Pläne sind deutlich leistungsfähiger im Vergleich zu denen des Verbrauchstarifs. [34]

4.3.8 Szenario 8: Azure Function und gRPC

Azure Functions unterstützen Queue-, Zeitgeber- und HTTP-Trigger [35]. Ein gRPC-Trigger für Azure Functions wird bislang nicht unterstützt, weshalb das geplante Szenario 8 nicht realisiert werden kann.

4.4 Definition der Szenarien mit temporal entkoppelter Kommunikation

Command-Services nach CQRS arbeiten besonders gut mit temporal entkoppelter Kommunikation über einen Message-Broker zusammen, da bei dem Absetzten eines Kommandos nicht unmittelbar eine Antwort mit dem Ergebnis benötigt wird. Stattdessen wartet der Client üblicherweise und fragt nach einer gewissen Zeit über das Lesemodell die Antwort auf das Kommando ab. Falls das Ergebnis zu dem entsprechenden Zeitpunkt noch nicht verfügbar ist, wartet der Client erneut und wiederholt den Prozess.

Für die Entwicklung des exemplarischen Command-Services wird als fachliche Basis der Warenkorb verwendet. Deshalb wird der Command-Service im Folgenden als Warenkorb-Command-

Service bezeichnet. Um die Funktionalitäten des Warenkorb-Command-Service weiter einzuschränken, wird außerdem lediglich die Operation des Hinzufügens von Produkten zu einem existierenden Warenkorb betrachtet.

Der Warenkorb-Command-Service wird auf der Basis des Event-Sourcing-Patterns modelliert, da dieser Ansatz in Bezug auf die Datenbank ohne Transaktionssperren bei dem Schreiben der Events auskommt und dementsprechend besser skaliert. Die finale Validierung des Kommandos wird so modelliert, dass sie bei der Projektion aller Events in eine materialisierte Warenkorbansicht in dem zum Warenkorb-Command-Service zugehörigen Query-Service stattfindet. Damit ist die einzige Aufgabe des Warenkorb-Command-Services die Speicherung der Kommandos in einem Event-Store. Es wäre zwar möglich den Event-Stream vor der Speicherung zu bereinigen, allerdings wird darauf zur Vereinfachung verzichtet.

Als Event-Store wird die NoSQL-Datenbank Azure Tables der Azure Cosmos DB verwendet. Diese verwendet den Kapazitätsmodi Serverless da dieser keine Kapazitätsplanung erfordert und in der Lage ist tausende Requests pro Sekunde zu verarbeiten [36]. Zudem eignet sich der serverlose Kapazitätsmodi, laut Dokumentation, gut für Azure Functions [36]. Der Nachteil der serverlosen Azure Tables ist, das sie zu dem Durchsatz und der Latenz der Datenbank keine Garantieversprechen machen [36]. Für die Kommunikation mit der Datenbank wird die Table API verwendet.

Als Message-Broker wird Azure Service Bus (PaaS) mit Queues verwendet. Um die Zustellung der Nachrichten zu garantieren wird die Anzahl der maximalen Zustellungen einer Nachricht auf das Maximum von 2000 gesetzt. Für Szenarien mit niedriger Last wird 1 GB als Kapazität der Queue gewählt, Szenarien mit mittlerer Last 3 GB und Szenarien mit hoher Last 5 GB. Die Wahl der Kapazität einer Queue ist abhängig von der Größe der Nachricht, dem Durchsatz der Worker-Instanzen und möglichen Lastspitzen eingehender Nachrichten. Die Azure Service Bus Queue verwendet außerdem die standardmäßig konfigurierte Nachrichtensperre von 30 Sekunden.

4.4.1 Szenario A: Azure Function

Szenario A verwendet serverlose Azure Functions (Verbrauchsplan) als Hosting-Umgebung. Die Azure Functions verwenden den Azure Service Bus Trigger. Dieser Trigger startet genau eine Azure Function für eine eingehende Nachricht in der Azure Service Bus Queue.

4.4.2 Szenario B: Azure App Service

Szenario B verwendet als Hosting-Umgebung den Azure App Service. Dieser wird analog zu Szenario 1 (siehe 4.3.1.) konfiguriert. Dies bedeutet, dass das Preismodell P1v3 mit zwei CPUs und 8 GB Arbeitsspeicher verwendet wird. Die Autoskalierung des Azure App Services wird deaktiviert.

4.4.3 Szenario C: Azure Container Instance

Szenario C verwendet als Hosting-Umgebung Azure Container Instances die analog zu Szenario 5 (siehe 4.3.5.) konfiguriert werden. Das Szenario verwendet somit ebenfalls zwei CPUs und 8 GB Arbeitsspeicher.

4.4.4 Szenario D: Azure VM

Szenario D verwendet eine virtuelle Maschine als Hosting-Umgebung. Diese wird analog zu Szenario 3 (siehe 4.3.3.) konfiguriert und besitzt somit zwei CPUs und 8 GB Arbeitsspeicher. Verwendet wird ebenfalls die Standard_D2ds_v5 VM der Ddsv5-Serie.

4.5 Definition der Auslastungsszenarien

Um in der Simulation möglichst realistische Gegebenheiten zu schaffen, ist es notwendig unterschiedliche Auslastungen der Services bei der Betrachtung der CO_2-äquivalenten Emissionen zu berücksichtigen, da der Energieverbrauch der Hardware in Verbindung mit der Auslastung der Ressourcen steht. Die unterschiedlichen Auslastungen der Services werden im Folgenden als Auslastungsszenarien definiert.

Die Auslastungen der Services werden durch Anfragen, im Kontext der temporal gekoppelten Kommunikation, bzw. Nachrichten, im Kontext der temporal entkoppelten Kommunikation, pro Sekunde modelliert. Es wird davon ausgegangen, dass sich die Last über die gesamte Testzeitspanne konstant verhält, um einen vergleichbaren Richtwert der Last und der damit einhergehenden CO_2-Emissionen zu erzeugen. Die betrachtete Testzeitspanne wird auf zwei Stunden gesetzt.

Insgesamt werden drei Last-Szenarien definiert. In jedem Lastszenario wird angenommen, dass eine Person alle vier Sekunden eine Anfrage bzw. Nachricht absendet. Im Folgenden werden sowohl Nachrichten als auch Anfragen zur Vereinfachung einheitlich als Anfragen bezeichnet.

In dem ersten Lastszenario wird von 10 gleichzeitig aktiven Personen ausgegangen, die den Web-Shop besuchen. Dies bedeutet, dass pro Sekunde 2,5 Anfragen versendet werden. Über einen Zeitraum von einer Minute entspricht dies 150 Anfragen und über den Zeitraum von einer Stunde 9.000 Anfragen.

In dem zweiten Lastszenario wird von 100 gleichzeitig aktiven Personen ausgegangen. Dies entspricht 25 Anfragen pro Sekunde, wenn wiederum davon ausgegangen wird, dass eine Person alle vier Sekunden eine Anfrage absendet. Über den Zeitraum von einer Minute entspricht dies 1.500 Anfragen und über den Zeitraum von einer Stunde 90.000 Anfragen.

In dem dritten Lastszenario wird von 1000 gleichzeitig aktiven Personen ausgegangen. Dies bedeutet, dass pro Sekunde 250 Anfragen an den entsprechenden Service gesendet werden. Pro Minute entspricht dies 15.000 Anfragen und pro Stunde 900.000 Anfragen.

5 Präzisierung der Methodik für die Azure Cloud

Nachdem die Rahmenbedingungen der Simulation festgelegt sind, wird im Folgenden die Methodik für die Erfassung und Berechnung der CO_2-äquivalenten Emissionen beschrieben.

Die Cloud Carbon Footprint-Software berechnet eine ganzheitliche Übersicht der CO_2-Emissionen pro Azure Abonnement. Dabei ist es möglich, die Emissionsdaten nach verwendetem Service zu gruppieren. Es ist bspw. möglich, die Emissionen aller Azure App Services einzusehen. Eine detaillierte Ansicht der Emissionsdaten pro Service bzw. pro Hosting-Umgebung ist jedoch nicht möglich, obwohl die Daten der Azure Consumption API pro Dienstleistung gruppiert sind und dies dementsprechend zulassen würden (siehe 3.3.2.).

Ein weiteres Problem ist, dass die Abrechnung der Dienstleistungen über die gesamte Dauer der Provisionierung erfolgt. Die definierten Szenarien haben jedoch eine festgelegte Laufzeit von genau zwei Stunden pro Szenario. Da die Provisionierung jedoch vor der Ausführung eines Szenarios erfolgt und das Deployment der Szenarien Zeit in Anspruch nimmt, werden die Ergebnisse

verfälscht. Das bedeutet, dass je nach Dienstleistung deutlich mehr verbrauchte Ressourcen abgerechnet werden können, wenn bspw. der Zeitpunkt der Provisionierung und Beginn des Szenarios weit auseinander liegt. Dasselbe gilt für die Zeitspanne zwischen der Beendigung der Ausführung eines Szenarios und der Deprovisionierung der Ressourcen.

Zusätzlich zu den beiden genannten Problemen kommt hinzu, dass die Cloud Carbon Footprint-Software für Microsoft Azure immer von einer konstanten vCPU-Auslastung von 50 % ausgeht, da eine Erfassung der vCPU-Auslastung noch nicht implementiert ist [10]. Durch die Verwendung der konstanten vCPU-Auslastung hätten die definierten Auslastungsszenarien bei der Datenerhebung durch die Cloud Carbon Footprint-Software keinerlei Einfluss. Es würde lediglich die Ausführungsdauer eines Szenarios ins Gewicht fallen. Die Messergebnisse wären damit nicht geeignet für den Vergleich der Hosting-Umgebungen und Kommunikationsmuster bzw. Kommunikationstechnologien, da diese die vorhandene Ressource potenziell unterschiedlich auslasten könnten.

Aus diesen drei genannten Aspekten eignet sich die Cloud Carbon Footprint Software ohne Anpassungen nicht für die CO_2-Emissionsdatenerfassung der einzelnen Szenarien. Dementsprechend ist eine Präzisierung der Methodik nach Cloud Carbon Footprint für die Microsoft Azure-Cloud notwendig, welche sowohl Ausführungsdauer als auch CPU-Auslastung mit einbezieht.

5.1 Datenerfassung mit Application Insights

Application Insights ist eine Funktionalität von Azure Monitor für die Überwachung der Leistungen von Web-Anwendungen [37]. Zu den überwachten Leistungen zählt unter anderem die prozentuale CPU-Auslastung einer Anwendung. Damit stellt Application Insights die erforderlichen Daten für die notwendige Präzisierung der Methodik nach Cloud Carbon Footprint zur Verfügung. Neben den Daten, welche für die CO_2-Emissionsdatenerfassung notwendig sind, bietet Application Insights die Möglichkeit weitere Daten zu Erfassen, die für die Bewertung der Benutzbarkeit bzw. Performance einer Anwendung verwendet werden können.

Application Insights basiert darauf, dass die Web-Anwendung Telemetriedaten an Application Insights Ressource in der Microsoft Azure Cloud sendet. Die Application Insights Ressource sammelt die Telemetriedaten und stellt sie über unterschiedliche Wege zur Verfügung. [37]

Grundsätzlich gibt es zwei unterschiedliche Wege, um Application Insights in einem Webdienst zu aktivieren. Entweder wird in der Hosting-Umgebung ein Application Insights Agent installiert, welcher die Web-Anwendung überwacht [37]. Dies erfordert keine Änderungen an dem Quellcode einer Anwendung. Ein Application Insights-Agent kann entweder manuell installiert werden oder kann optional, im Falle von Azure-VM-Skalierungsgruppen, Azure App Services, Azure Functions und Azure Cloud Services bei der Bereitstellung der Ressource aktiviert werden [38]. Alternativ wird Application Insights über die Einbindung eines Software Development Kits (SDKs) im Quellcode der Anwendung aktiviert. SDK unterstützte Programmiersprachen sind C#/VB (.NET), Java, JavaScript und Python [38]. In diesem Fall muss jedoch ein in der Application Insights-Ressource generierter Instrumentierungsschlüssel manuell in der Web-Anwendung angegeben werden. [37]

Application Insights ermöglicht insbesondere die Überwachung der Systemleistungsindikatoren von Linux- oder Windows-Servern als protokollbasierte Metriken. Dazu zählen bspw. die Leistungsindikatoren CPU, Arbeitsspeicher und Netzwerkauslastung. Die Systemleistungsindikatoren werden je nach Hosting-Umgebung entweder direkt ausgelesen oder über Umgebungsvariablen bereitgestellt. Direkt ausgelesen werden sie, wenn die Anwendung auf einer virtuellen Maschine ausgeführt wird und der Administratorzugriff möglich ist. Über die Umgebungsvariablen werden die Systemleistungsindikatoren bereitgestellt, wenn die zugrundeliegende Anwendung in einer isolierten Hosting-Umgebung wie bspw. Azure Container Instances, Azure App Services oder Azure Functions ausgeführt werden. [39]

Zu den standardmäßig verfügbaren Systemleistungsindikatoren zählen die folgenden Indikatoren:

- Prozessorzeit (performanceCounters/processorCpuPercentage)
- Prozess-CPU (performanceCounters/processCpuPercentage)
- Private Bytes des Prozesses (performanceCounters/processPrivateBytes)

Der Systemleistungsindikator Prozess-CPU gibt an, wie viel der gesamt verfügbaren Prozessorleistung von dem überwachten Prozess genutzt wird. Dies bedeutet, dass der mögliche Wertebereich des Systemleistungsindikators in Abgrenzbarkeit zur Anzahl der CPU-Kerne einer Instanz steht. Der Wertebereich liegt damit zwischen 0 und $100 \cdot n$, wobei n für die Anzahl der verfügbaren vCPU-Kerne steht. Eine alternative Darstellung des Systemleistungsindikators ist die normalisier-

te Prozess-CPU-Zeit. Hier besteht die Annahme, dass diese sich durch den verwendeten Schlüssel *% Processor Time Normalized* erkennen lässt. Die Dokumentation beinhaltet dazu jedoch keine detaillierten Informationen. Der Wertebereich der alternativen Darstellung liegt zwischen 0 und 100, da durch die Anzahl der verfügbaren CPU-Kerne einer Instanz geteilt wird. [40]

Der Systemleistungsindikator Prozessorzeit gibt die gesamte CPU-Auslastung durch alle Prozesse, die auf der überwachten Instanz ausgeführt werden an [40]. Die Dokumentation beinhaltet keinerlei Informationen dazu, in welchem Wertebereich sich der Systemleistungsindikator befindet. Erste Beobachtungen zeigen jedoch, dass der Wert grundsätzlich das Doppelte des Prozess-CPU-Leistungsindikators ist. Da in dem Test eine vCPU mit zwei Kernen verwendet wird und es sich bei dem Prozess-CPU-Systemleistungsindikators um einen normalisierten Wert handeln muss, ist davon auszugehen, dass der Prozessorzeit-Systemleistungsindikator die Summe aller prozentualen vCPU-Kernauslastungen darstellt. Insbesondere, weil der Wert während der Beobachtung die 100 Prozent in dem Test überschreitet. Dies wiederum deutet darauf hin, dass die zuvor getroffene Annahme korrekt ist.

Der Leistungsindikator Private Bytes des Prozesses gibt an, wie viel Arbeitsspeicher von dem überwachten Prozess verwendet wird bzw. reserviert ist. [40]

Alle Leistungsindikatoren besitzen die Dimension *cloud_RoleInstance*, die die Identität der Hosting-Instanz angibt, auf dem ihre Anwendung ausgeführt wird [39]. Dies ist notwendig, da bei einer Autoskalierung potenziell mehrere Instanzen Systemleistungsindikatoren an die Application Insights-Ressource senden.

Neben den Systemleistungsindikatoren, welche für die Umrechnung in CO_2-Emissionen notwendig sind, werden weitere Daten für die Bewertung der Benutzbarkeit eines Szenarios benötigt. Dafür bietet es sich an, die Dauer der Bearbeitung eines Requests bzw. einer Nachricht als Bewertungsmetrik zu verwenden. Je nach Art der Kommunikation ist dabei ein unterschiedlicher Ansatz für die Extraktion der Daten notwendig. Bei der temporal gekoppelten Kommunikation ist die Dauer von dem Beginn der Anfrage bis hin zum Eintreffen der Antwort entscheidend, während bei der temporal entkoppelten Kommunikation der Zeitpunkt zwischen dem Absenden der Nachricht und der Bearbeitung ausschlaggebend ist. Daher müssen bei der temporal gekoppelten Kommunikation (Query-Services) die Daten auf der Client-Seite erfasst werden. Die Ausführungsdauer eines

temporal entkoppelten Szenarios (Command-Services) lässt sich dagegen nicht Client-seitig ermitteln, da die Kommunikationsverbindung unmittelbar nach dem erfolgreichen Versenden einer Nachricht an einen Message-Broker getrennt wird. Stattdessen muss die Ausführungsdauer über den Startzeitpunkt, welcher Teil der Nachricht ist und den Endzeitpunkt, welcher Server-seitig erfasst wird, berechnet werden. Das Ergebnis der Berechnung wird dann als eigene protokollbasierte Metrik über Application Insights erfasst.

Weitere verfügbare protokollbasierte Metriken sind Servermetriken, wie Abhängigkeitsaufrufe, Dauer der Abhängigkeit, Serverantwortzeit und Serveranforderungen [40]. Auch diese Metriken sollten für die Bewertung der Performance bzw. Usability miteinbezogen werden. Abhängigkeiten sind externe Dienstleistung, wie bspw. eine Datenbank oder andere Services. Automatisch erfasste Abhängigkeiten sind externen oder lokale HTTP/HTTPS-Aufrufe, SQL-Aufrufe, Azure-Speicher-Aufrufe (Azure Blob Storage, Azure Tables oder Azure Queues), Anfragen über Azure EventHub-Client-SDK und Azure ServiceBus-Client-SDK sowie Anfragen an eine Azure Cosmos DB insofern eine HTTP- bzw. HTTPS-basierte Kommunikation verwendet wird [41].

Die Extraktion der protokollbasierten Metriken erfolgt mithilfe der Datenabfragesprache Kusto. Die konkreten Abfragen werden im Folgenden detailliert beschrieben.

5.2 Datenentnahme aus Application Insights

Die Kusto Query Language (KQL) ist die Datenabfragesprache für protokollbasierte Metriken, die über Application Insights erfasst werden. Die Abfragesprache ähnelt teilweise Structured Query Language (SQL) und verwendet ebenfalls eine SQL-ähnliche hierarchische Organisation der Daten durch Datenbanken, Tabellen und Spalten. [42]

Eine Problematik bei der Extraktion der Daten aus Application Insights ist, dass protokollbasierte Metriken in einem Hintergrund-Thread nicht blockierend gesammelt, aggregiert und schlussendlich an die Application Insights-Ressource in der Microsoft Azure-Cloud versendet werden. Dies hat zur Folge, dass die Zeitstempel der aggregierten Datenpunkte minimal von der tatsächlichen Zeit abweichen können. Die Annahme ist, dass der Versatz der Zeit die zwei Minuten nicht überschreitet. Dies kann u. a. dadurch begründet werden, dass die Datenpunkte in einem einminütigen Intervall versendet werden. Um nun das Zeitintervall finden zu können, in welchem ein Szenario

gelaufen ist, muss der Startzeitpunkt bekannt sein. Der Startzeitpunkt wird durch das Schreiben in eine Datei in der Client-Hosting-Umgebung in der UTC-Zeitzone festgehalten, da die Zeitstempel der protokollbasierten Metriken ebenfalls in UTC gespeichert werden. Durch den bereits genannten möglichen zeitlichen Versatz wird das Zeitintervall bei dem Auslesen der Application Insights-Daten um zwei Minuten verlängert. Da es leichte Unterschiede bei der Erfassung der Application Insights-Daten von Query- (temporal gekoppelt) und Command-Services (temporal entkoppelt) gibt, werden diese im Folgenden teilweise getrennt beschrieben.

Zunächst wird die Kusto Abfrage für das Auslesen der Query-Service-Szenarien betrachtet. Da diese in einzelne Funktionen aufgeteilt ist, werden die einzelnen Funktionen zudem getrennt betrachtet.

```
let query_compute_by_instance = (begin: datetime, ts: timespan) {
    let interval = performanceCounters
        | where timestamp >= begin and timestamp <= begin + ts;

    let processorTime = interval
        | where counter == "% Processor Time"
        | project timestamp, cloud_RoleInstance, ProcessorTime=value;

    let processorTimeNormalized = interval
        | where counter == "% Processor Time Normalized"
        | project timestamp, cloud_RoleInstance, ProcessorTimeNormalized=value;

    processorTime
        | join kind=inner processorTimeNormalized on timestamp, cloud_RoleInstance
        | project timestamp, cloud_RoleInstance, ProcessorTime, ProcessorTimeNormalized
        | extend CPU = ProcessorTime / iff(ProcessorTimeNormalized == 0, 99999,
↪   ProcessorTimeNormalized)
        | summarize
            average_processor_time = avg(ProcessorTime),
            average_processor_time_normalized = avg(ProcessorTimeNormalized),
            cpu_count=round(max(CPU), 0),
            instance_begin = min(timestamp),
            instance_end= max(timestamp) by cloud_RoleInstance
        | extend instance_duration = instance_end - instance_begin
};
```

Abbildung 3: Kusto: Abfrage der CPU-Auslastung

Zunächst wird die durchschnittliche vCPU-Auslastung, die durchschnittliche normalisierte vCPU-Auslastung und die Anzahl der vCPU-Kerne gruppiert durch die Hosting-Instanzen eines Szenarios ausgelesen (siehe Abb. 3, Seite 28). Die Abfrage wird als Funktion mit dem Namen *query_-compute_by_instance* realisiert. Die Parameter der Funktion sind der Startzeitpunkt eines Szenarios und die Ausführungsdauer. Zunächst werden die Datenpunkte der Tabelle *performanceCounters* (Systemleistungsindikatoren) auf das Zeitintervall, in dem das Szenario gelaufen ist, limitiert und in der Variable *interval* gespeichert (Z. 2, Abb. 3). Anschließend werden die Zeilen, welche die vCPU-Auslastung und normalisierte vCPU-Auslastung beinhalten in die Variablen *processorTime* bzw. *processorTimeNormalized* gespeichert. Dabei werden nur die Spalten mit dem Zeitstempel, Instanzname und Wert selektiert. Die beiden Tabellen aus den Variablen *processorTime* und *processorTimeNormalized* werden über eine inneren Join über den Instanznamen (*cloud_RoleInstance*) zusammengeführt (Z. 13, 14, Abb. 3). Da der Join-Operator die Spalten mit den Instanznamen dupliziert, wird mit dem *project*-Operator eine dieser Spalten entfernt.

Durch die Annahme, dass der Leistungsindikator Prozess-CPU (*% Processor Time Normalized*) die normalisierte Variante des Leistungsindikators ist, also die normalisierte vCPU-Auslastung (vCPU-Auslastung pro vCPU-Kern), und der Tatsache, dass in einem Testszenario mit zwei vCPU-Kernen der Leistungsindikator Prozess-CPU zu einem Zeitpunkt immer exakt die Hälfte des Leistungsindikators Prozessorzeit (*% Processor Time*) ist, lässt sich die Anzahl der vCPU-Kerne durch die Division des Leistungsindikators Prozess-CPU durch den Leistungsindikator Prozessorzeit ermitteln (Z. 16, Abb. 3). Dies ist jedoch nur für die serverlosen Azure Functions von Bedeutung, da in allen anderen Szenarien die Anzahl der vCPU-Kerne bekannt bzw. definiert ist. Eine Alternative Methodik für die Ermittlung der vCPU-Kerne einer Hosting-Instanz scheint es für die serverlosen Azure Functions bisher nicht zu geben.

Da die Methodik der Cloud Carbon Footprint-Software für die Umrechnung der Nutzung in Energie auf der durchschnittlichen vCPU-Auslastung über die gesamte Ausführungsdauer basiert, wird der Durchschnitt mithilfe der Funktion *avg* gebildet (Z. 18, 19, Abb. 3). Außerdem wird das Maximum der vCPU-Kerne gesucht, da die Annahme besteht, dass sie die Anzahl der vCPU-Kerne im Verlauf der Ausführung nicht ändert. Notwendig ist dies, da zuvor für die Berechnung der Variable *CPU* im Falle der Division durch null, der Divisor durch eine große Zahl ersetzt wird, um zum einen keinen Fehler hervorzurufen und zum anderen keine Überanpassung zu verursachen

(Z. 16, Abb. 3). Für die Szenarien ohne fest provisionierter Hosting-Instanzen (bspw. Serverless Azure Functions) wird der Zeitstempel mit minimalem und maximalem Zeitwert innerhalb des Intervalls gesucht (Z. 21, 22, Abb. 3). Auf der Basis dieser Zeitstempel wird dann die Laufzeit der Hosting-Instanzen *instance_duration* durch die Differenzbildung ermittelt. Hierbei wird die Annahme getroffen, dass eine Hosting-Instanz so lange aktiv ist, wie Log-Einträge an die Application Insights-Ressource versendet werden.

```
let query_ram_by_instance = (begin: datetime, ts: timespan) {
    let counters = performanceCounters
        | where timestamp >= begin and timestamp <= begin + ts;

    let ram_available = counters
        | where name == "Available Bytes"
        | summarize average_ram_available_bytes = avg(value) by cloud_RoleInstance;

    let ram_used = counters
        | where name == "Private Bytes"
        | summarize average_ram_used_bytes = avg(value) by cloud_RoleInstance;

    ram_used
        | join kind=fullouter ram_available on cloud_RoleInstance
        | project-away cloud_RoleInstance1
};
```

Abbildung 4: Kusto: Abfrage der Arbeitsspeicherauslastung

Über die Funktion *query_ram_by_instance* wird der verfügbare Arbeitsspeicher und der von einer Hosting-Instanz reservierte Arbeitsspeicher ermittelt (siehe Abb. 4, Seite 30). Die Funktion erhält wie zuvor den Startzeitpunkt und die Ausführungsdauer als Parameter, um die Datenpunkte auf das Zeitintervall der Ausführung zu begrenzen. Anschließend wird innerhalb des Zeitintervalls der durchschnittlich verfügbare Arbeitsspeicher sowie der durchschnittlich verwendete Arbeitsspeicher pro Hosting-Instanz berechnet und in die Variablen *ram_available* bzw. *ram_used* gespeichert. Zuletzt werden die beiden Tabellen mithilfe eines äußeren Verbundes zusammengeführt, um im Falle von der Abwesenheit einer der beiden Werte, den jeweils anderen Wert nicht zu verlieren. Dies ist bspw. bei serverlosen Azure Functions der Fall, da dort kein Wert für den verfügbaren Arbeitsspeicher vorhanden ist.

```
let query_io_by_instance = (begin: datetime, ts: timespan) {
    performanceCounters
      | where timestamp >= begin and timestamp <= begin + ts
      | where name == "IO Data Bytes/sec"
      | summarize avg_io_bytes_sec = avg(value) by cloud_RoleInstance;
};
```

Abbildung 5: Kusto: Abfrage der IO-Auslastung

Über die Funktion *query_io_by_instance* wird der durchschnittlich eingehende und ausgehende Datenverkehr pro Hosting-Instanz ermittelt (siehe Abb. 5, Seite 31).

```
let query_requests_data_per_instance = (operation_Name: string, begin: datetime, ts:
↪ timespan) {
    let req_over_timespan = requests
        | where timestamp >= begin and timestamp <= begin + ts
        | where operation_Name == operation_Name;

    req_over_timespan
        | summarize (
            request_percentile_duration_25,
            request_percentile_duration_50,
            request_percentile_duration_75,
            request_percentile_duration_90,
            request_percentile_duration_95,
            request_percentile_duration_99) = percentiles(duration, 25, 50, 75, 90, 95,
↪ 99),
            request_min_duration = min(duration),
            request_average_duration = avg(duration),
            request_max_duration=max(duration),
            incoming_request_count = sum(itemCount)
            by cloud_RoleInstance;
};
```

Abbildung 6: Kusto: Abfrage der Request-Auslastung

Über die Funktion *query_requests_data_per_instance* wird die Anzahl der Anfragen sowie deren Ausführungsdauer pro Hosting-Instanz ausgelesen (siehe Abb. 6, Seite 31). Die Quelle der Daten ist die Tabelle *requests* der protokollbasierten Metriken. Für Services die temporal gekoppelte Kommunikation verwenden ist die Ausführungsdauer die Zeit, welche für die Bearbeitung eines

HTTP- bzw. HTTPS-Requests benötigt wird und für Service, die temporal entkoppelte Kommunikation verwenden, ist die Ausführungsdauer die Zeit zwischen dem Eintreffen einer Nachricht bei einer Worker-Instanz und dem Abschluss der Nachricht. Pro Hosting-Instanz werden die p-Quantile der Ausführungsdauer $p = 25, p = 50, p = 75, p = 90, p = 95, p = 99$ mithilfe der Funktion *percentiles* berechnet und in den entsprechenden Variablen hinterlegt (Z. 8-13, Abb. 6). Die oberen Perzentile $p = 90, p = 95, p = 99$ eigenen sich um die Zuverlässigkeit eines Services bezüglich der Ausführungsdauer besser bewerten zu können. Neben den p-Quantilen wird die minimale, maximale und durchschnittliche Ausführungsdauer sowie die gesamte Anzahl der Requests pro Hosting-Instanz berechnet (Z. 14-17, Abb. 6).

```
let query_dependencies_per_instance = (opName: string, begin: datetime, ts: timespan) {
    dependencies
        | where timestamp >= begin and timestamp <= begin + ts
        | where type == "SQL"
        | summarize
            (dependency_percentile_duration_25,
                dependency_percentile_duration_50,
                dependency_percentile_duration_75,
                dependency_percentile_duration_90,
                dependency_percentile_duration_95,
                dependency_percentile_duration_99) = percentiles(duration, 25, 50, 75,
 ↪  90, 95, 99),
            dependency_min_duration = min(duration),
            dependency_average_duration = avg(duration),
            dependency_max_duration=max(duration),
            dependency_request_count = sum(itemCount)
            by cloud_RoleInstance
};
```

Abbildung 7: Kusto: Abfrage der Dauer von Abhängigkeiten

Analog zur Auswertung der Ausführungsdauer von Requests werden die Abhängigkeiten (Datenbanken) über die Funktion *query_dependencies_per_instance* pro Hosting-Instanz aus der Tabelle *dependencies* ausgelesen (siehe Abb. 7, Seite 32). Je nach verwendeter Datenbank ist der Typ *SQL* oder im Falle eines Azure Cosmos DB Tables *Microsoft.Tables*.

```
let query_all = (operation_Name: string, begin: datetime, ts: timespan) {
    query_compute_by_instance(begin, ts)
    | join kind=fullouter query_ram_by_instance(begin, ts) on cloud_RoleInstance
    | project-away cloud_RoleInstance1
    | join kind=fullouter query_io_by_instance(begin, ts) on cloud_RoleInstance
    | project-away cloud_RoleInstance1
    | join kind=fullouter query_requests_data_per_instance(operation_Name, begin, ts) on
    ↪ cloud_RoleInstance
    | project-away cloud_RoleInstance1
    | join kind=fullouter query_dependencies_per_instance(operation_Name, begin, ts) on
    ↪ cloud_RoleInstance
    | project-away cloud_RoleInstance1
};
```

Abbildung 8: Kusto: Kombination aller Abfragefunktionen

Die Funktion *query_all* kombiniert alle zuvor beschriebenen Funktionen über den Join-Operator (siehe Abb. 9, Seite 33). Das Ergebnis ist eine Tabelle die pro Hosting-Instanz zeilenweise die aggregierten Daten beinhaltet.

Für die gesamte Abfrage der Daten eines temporal gekoppelten Szenarios wird der Funktion *query_all* der Name der Service-Operation, der Startzeitpunkt des Szenarios sowie die Zeitspanne des Szenarios inklusive des optimistischem zweiminütigen Versatzes übergeben (siehe Abb. 9, Seite 33). Die Ergebnisse werden anschließend mithilfe der *Export to CSV* Funktion des Application Insight-Explorers persistiert.

```
query_all("QueryProducts", todatetime('2022-07-23 16:34:23.248'), timespan(2h) +
↪ timespan(2m))
```

Abbildung 9: Kusto: Ausführung aller Abfragen

Die Abfrage der Ausführungsdauer eines temporal entkoppelten Szenarios erfordert eine zusätzliche Funktion, da die entsprechenden Daten als eigene protokollbasierte Metrik erfasst werden (siehe 5.1.). Entsprechend wird die Funktion *query_duration_data_per_instance* definiert, welche analog zu den Abhängigkeiten oder der Server-seitigen Ausführungsdauer, die gesamte Abarbeitungs- bzw. Ausführungsdauer eines temporal entkoppelten Szenarios ausliest (siehe Abb. 10, Seite 34). Eigene protokollbasierte Metriken werden von Application Insights in der Tabelle *customMetrics*

gespeichert. Das Tabellenschema ähnelt dem Tabellenschema der Systemleistungsindikatoren und ermöglicht ebenfalls die Gruppierung nach Hosting-Instanz. Gespeichert wird die berechnete Ausführungsdauer in Millisekunden mit dem Namen *DurationMs*.

```
let query_duration_data_per_instance = (begin: datetime, ts: timespan) {
    let duration_over_timespan = customMetrics
        | where timestamp >= begin and timestamp <= begin + ts
        | where name == "DurationMs";

        duration_over_timespan
        | summarize (
            full_percentile_duration_25,
            full_percentile_duration_50,
            full_percentile_duration_75,
            full_percentile_duration_90,
            full_percentile_duration_95,
            full_percentile_duration_99) = percentiles(value, 25, 50, 75, 90, 95, 99),
            full_min_duration = min(value),
            full_average_duration = avg(value),
            full_max_duration=max(value),
            full_request_count = sum(valueCount)
            by cloud_RoleInstance;
};
```

Abbildung 10: Kusto: Abfrage der Bearbeitungsdauer

Analog zu den beschriebenen Funktionen werden weitere Funktionen implementiert, die nicht nach der Hosting-Instanz gruppieren, da insbesondere die p-Quantile nach der Datenentnahme nicht statistisch sinnvoll aggregierbar sind. Für die Analyse der Daten sind jedoch zur Vereinfachung der Visualisierung Box-Plots pro Szenario geplant, weshalb insbesondere einige p-Quantile benötigt werden. Zusätzlich zu den in den zuvor beschriebene Funktionen werden außerdem die p-Quantile $p-1$, $p-2$, $p-5$, $p-10$ und $p-98$ in dem ganzheitlichen Datensatz berücksichtigt. Auf die detaillierte Beschreibung der Funktionen wird jedoch verzichtet, da sie sich, wie bereits erwähnt, sehr stark den bereits beschriebenen Funktionen ähneln.

Zusammengefasst ermöglicht Application Insight die Entnahme der folgenden Daten als CSV-Tabelle:

1. Systemleistungsindikatoren (relevant für die Berechnung der CO_2-Emissionen)

   a) Durchschnittliche Arbeitsspeicherauslastung
   b) Durchschnittliche CPU-Auslastung
   c) Absolute Anzahl an CPUs (relevant bei Serverless),
   d) Durchschnittlich verfügbarer Arbeitsspeicher (teilweise)
   e) Durchschnittliche Netzwerkauslastung (teilweise)
2. Datenbanklatenzen (durch Transitivität relevant für die Bewertung der Performance und Usability)
   a) Unterschiedliche p-Quantile (pro Hosting-Instanz und ganzheitlich)
   b) Durchschnittliche Dauer (pro Hosting-Instanz und ganzheitlich)
   c) Maximale Dauer (pro Hosting-Instanz und ganzheitlich)
   d) Minimale Dauer (pro Hosting-Instanz und ganzheitlich)
   e) Stichprobengröße (pro Hosting-Instanz und ganzheitlich)
3. Server-seitige Bearbeitungsdauer von Requests (durch Transitivität relevant für die Bewertung der Performance und Usability sowie im Falle der serverlosen Azure Function für die Abschätzung der Kosten)
   a) Unterschiedliche p-Quantile (pro Hosting-Instanz und ganzheitlich)
   b) Durchschnittliche Dauer (pro Hosting-Instanz und ganzheitlich)
   c) Maximale Dauer (pro Hosting-Instanz und ganzheitlich)
   d) Minimale Dauer (pro Hosting-Instanz und ganzheitlich)
   e) Stichprobengröße (pro Hosting-Instanz und ganzheitlich)
4. Bearbeitungsdauer von temporal entkoppelten Nachrichten über eine eigens definierte Metrik (relevant für die Bewertung der Performance und Usability)
   a) Unterschiedliche p-Quantile (pro Hosting-Instanz und ganzheitlich)
   b) Durchschnittliche Dauer (pro Hosting-Instanz und ganzheitlich)
   c) Maximale Dauer (pro Hosting-Instanz und ganzheitlich)
   d) Minimale Dauer (pro Hosting-Instanz und ganzheitlich)
   e) Stichprobengröße (pro Hosting-Instanz und ganzheitlich)

Die gesamte Ausführungsdauer einzelner Requests der Szenarien mit temporal gekoppelter Kommunikation (Query-Services) wird Client-seitig über die Bibliothek NBomber erfasst und ist somit nicht in der Tabelle enthalten.

Die durch NBomber erfasste Metriken sind insbesondere für die Bewertung der Performance und Usability relevant. Die erfassten Metriken sind:

1. Durchsatz (Anfragen pro Sekunde)

2. Minimale, maximale und Durchschnittliche Dauer der Anfragen sowie die Standardabweichung
3. Unterschiedliche p-Quantile ($p-50$, $p-75$, $p-95$ und $p-99$)
4. Anzahl der erfolgreichen und fehlgeschlagenen Anfragen
5. Menge an übertragenen Daten (in Kilobyte)

5.3 Anpassung der Energieberechnung

Mithilfe der über Application Insights erfassten Daten pro Hosting-Instanz ist es möglich, die in der Cloud Carbon Footprint-Software fehlende vCPU-Auslastung in der überarbeiteten Methodik mit einzubeziehen. Die Energieberechnung (siehe 3.3.4.) wird außerdem pro Hosting-Instanz durchgeführt und nicht, wie in der Software Cloud Carbon Footprint pro Dienst. Damit beachtet die überarbeitete Methodik zusätzlich die unterschiedlichen Auslastungen der einzelnen Hosting-Instanzen. Für die Berechnung des Endergebnisses werden die einzeln berechneten Wattstunden pro Service aufsummiert. Dies ist jedoch nur für die Szenarien mit der serverlosen Hosting-Umgebung Azure Function relevant, da diese eine automatische Skalierung der Hosting-Instanzen besitzen und dementsprechend mehrere Hosting-Instanzen verwenden können.

Bei den Szenarien mit den Hosting-Umgebungen Azure App Service, Azure Container Instance und Azure VMs wird die Laufzeit der Szenarien, welche zwei Stunden entspricht, mit der Anzahl der Konfigurieren vCPUs multipliziert, um die vCPU-Stunden zu berechnen. Da die genannten Hosting-Umgebungen alle zwei vCPUs besitzen, ergibt sich damit die konstante vCPU-Zeit von zwei vCPU-Stunden. Die vCPU-Stunden werden dementsprechend nicht, wie es bei der Cloud Carbon Footprint-Software der Fall ist, über die Verbrauchsdaten erfasst (siehe 3.3.2.).

Bei den Szenarien, welche eine automatische Skalierung besitzen (Azure Functions) wird die Anzahl der vCPUs und die vCPU-Zeit pro Hosting-Instanz über die Daten, welche mithilfe von Application Insights erfasst werden, berechnet. Die Annahme ist hier, dass zum Zeitpunkt des ersten Auftretens eines Zeitstempels einer spezifischen Hosting-Instant, diese für die Bearbeitung von Anfragen bzw. Nachrichten temporär provisioniert wird. Außerdem wird davon ausgegangen, dass das letzte Auftreten eines solchen Zeitstempels in den protokollbasierten Metriken das Ende der Ausführung einer solchen Hosting-Instanz markiert. Dementsprechend wird die Ausführungszeit

einer Hosting-Instanz über die Zeitspanne zwischen den beiden genannten Zeitstempeln berechnet.

Die vCPU-Stunden werden durch die Multiplikation der Ausführungszeit einer Hosting-Instanz t_h mit der berechneten oder definierten Anzahl der vCPUs pro Hosting-Instanz $|\text{vCPU}_h|$ berechnet.

$$\text{vCPU}h_h = |\text{vCPU}_h| \cdot t_h \tag{5}$$

Die Berechnung der Energie erfolgt durch die Summe der Energieberechnungen pro Hosting-Instanz, wobei n die Anzahl der erfassten Hosting-Instanzen ist (6). Es wird die durchschnittliche prozentuale vCPU-Auslastung einer Hosting-Instanz $\overline{\text{vCPU}\%}_h$ und die vCPU-Stunden einer Hosting-Instanz $\text{vCPU}h_h$ miteinbezogen.

$$Wh = \sum_{h=1}^{h=n} W_{\text{min.}} + \overline{\text{vCPU}\%}_h \cdot (W_{\text{max.}} - W_{\text{min.}}) \cdot \text{vCPU}h_h \tag{6}$$

Ein alternativer und auch grundsätzlich besserer Ansatz für die Energieberechnung wäre die Integralbildung über die Ausführungszeit mit der prozentualen vCPU-Auslastung als diskrete Funktion statt der Methodik nach Cloud Carbon Footprint, welche die durchschnittliche prozentuale vCPU-Auslastung über den gesamten Zeitraum der Ausführung verwendet. Da die definierten Auslastungsszenarien jedoch konstant über die definierte Laufzeit von zwei Stunden sind, kann die von der Cloud Carbon Footprint adaptierte Methodik als hinreichend genau betrachtet werden.

Dienste, wie bspw. Datenbanken, werden nicht in die Energieberechnung miteinbezogen insbesondere, da für die Azure Cosmos DB die verfügbare Datenlage, aufgrund der fehlenden Transparenz der Microsoft Azure Cloud, zu gering ist, um eine sinnvolle Abschätzung treffen zu können. Auch der produzierte Traffic, welcher durch die Anwendung verursacht wird, wird nicht in die Energieberechnung miteinbezogen, da zum einen die benötigte Energie für den Datentransport innerhalb des Rechenzentrums vernachlässigbar ist [11] und zum anderen der Traffic außerhalb des Rechenzentrums im Falle eines Web-Shops, durch die einheitliche Kommunikation per HTTP, pro Auslastungsszenario konstant wäre und demnach keine Rolle bei der Bewertung der Szenarien spielen würde.

6 Implementierung der Simulation

Alle Szenarien und die Client-Anwendungen werden als C# .NET 6.0 Anwendung implementiert. Für Hosting-Umgebungen, welche nicht automatisch über einen Application Insights Agent verfügen (Azure VM und Azure Container Instance) wird die Quellcode-basierte Implementierung von Application Insights verwendet.

6.1 Implementierung der Szenarien mit temporal gekoppelter Kommunikation

Für die Szenarien mit temporal gekoppelter Kommunikation, welche eine SQL-Datenbank verwenden, wird das Framework EntityFrameworkCore für die Kommunikation mit der Datenbank verwendet.

Die Szenarien, welche eine Web-API als Kommunikationstechnologie bereitstellen, verwenden die durch AspNetCore bereitgestellten API-Controller. Diese serialisieren bei einer Anfrage durch die Client-Anwendung eine Liste der definierten des C# Records *ProductItem* (siehe Abb. 11, Seite 38) als JSON-Objekt.

```
public record ProductItem(
    int AmountAvailable,
    decimal EuroPrice,
    string? ImageSource,
    Guid ProductId,
    string ProductName
);
```

Abbildung 11: C# ProductItem

Szenarien, welche gRPC als Kommunikationstechnologie verwenden, stellen dieselben Daten über einen gRPC-Service (siehe Abb. 12, Seite 39) bereit.

```
service ProductCatalog {
    rpc QueryProducts (ProductsRequest) returns (ProductsResponse);
}
message ProductsRequest {
}
message ProductsResponse {
    repeated ProductItem products = 1;
}
message ProductItem {
    string productId = 1;
    string productName = 2;
    string imageSource = 3;
    DecimalValue euroPrice = 4;
    int32 amountAvailable = 6;
}
message DecimalValue {
    int64 units = 1;
    sfixed32 nanos = 2;
}
```

Abbildung 12: Protocol Buffers ProductCatalog-Service

Alle Szenarien verwenden dieselbe Geschäftslogik, die Produkte aus der SQL-Datenbank lädt und für diese eine einheitliche Preisberechnung durchführt. Die Szenarien unterschieden sich dementsprechend nur durch die Kommunikationstechnologie. Um eine realistische Situation zu schaffen, werden bei einer Anfrage jeweils 100 Produkte aus der Datenbank geladen und zurückgegeben.

6.2 Implementierung der Szenarien mit temporal entkoppelter Kommunikation

Bei den Szenarien mit temporal entkoppelter Kommunikation, welche die Azure Cosmos DB verwenden, wird das Azure Cosmos DB SDK für die Kommunikation mit der Datenbank verwendet. Die Szenarien verwenden alle eine einheitliche Geschäftslogik, die Nachrichten über den Azure Service Bus entgegennimmt und diese in der Azure Cosmos DB persistiert. Eingehende Nachrichten werden über ein geteilten Contract (siehe Abb. 13, Seite 40) per JSON serialisiert.

```
public class BasketAddedItem {
    public Guid CorrelationId { get; set; }
    public Guid ItemId { get; set; }
    public Guid BasketId { get; set; }
    public int Amount { get; set; }
}
```

Abbildung 13: C# BasketAddedItem

Um die gesamte Bearbeitungsdauer einer Nachricht zu erfassen, wird die durch den in der Nachricht enthaltenen Zeitstempel berechnete Bearbeitungsdauer an die Application Insights-Ressource gesendet.

Mit der Ausnahme der Szenarien mit Azure Function Hosting-Umgebung, verwenden alle Szenarien mit temporal entkoppelter Kommunikation die Klasse *ServiceBusProcessor*, um Nachrichten des Message-Brokers zu empfangen. Azure Functions verwenden statt dem *ServiceBusProcessor* den *ServiceBusTrigger* für das Empfangen von Nachrichten des Message-Brokers.

6.3 Erzeugung der Last

Für die Erzeugung der Last wird eine verhältnismäßig große Azure VM des Typs *Standard_B2s* der B-Serie verwendet. Diese besitzt 2 vCPUs und 4 GB Arbeitsspeicher. Damit wird garantiert, dass die Client-Hosting-Umgebung kein limitierender Faktor bei der Ausführung der Szenarien darstellt.

Für die Implementierung der Auslastungsszenarien wird die C# Bibliothek NBomber verwendet. Die Bibliothek übernimmt die Thread-Verwaltung und Parallelisierung, um den definierten Durchsatz zu garantieren. Da die Service übergreifende Kommunikation simuliert werden soll, wird die maximale Parallelisierung auf 10 Threads beschränkt. Die Bibliothek erfasst bei der Ausführung die Latenzen und den Durchsatz. Die erfassten Daten werden nach der Ausführung in eine CSV-Datei geschrieben.

Insgesamt werden drei verschiedene Clients implementiert. Einen für die Kommunikation mittels gRPC, einen für die Kommunikation mit einer klassischen Web-API per HTTP und ein Client, welcher Nachrichten an den Azure Service Bus Message-Broker sendet.

7 Herleitung der Emissionsfaktoren

Die durch die Szenarien und Auslastungsszenarien beeinflussten Eingangsparameter der CO_2-äquivalenten Emissionsberechnung sind die innerhalb der Ausführungszeit verursachten vCPU-Stunden und Wattstunden.

Die angepasste Formel für die Energieberechnung (Wattstunden) (siehe 5.3.) benötigt die minimale und maximale Leistungsaufnahme der zugrundeliegenden Prozessormikroarchitektur. Diese Koeffizienten werden wie folgt gewählt. Für die Hosting-Umgebungen Azure App Service, Azure Container Instance und serverlose Azure Function (Verbrauchsplan) ist es aufgrund der fehlenden Transparenz der Azure-Cloud nicht möglich die zugrundeliegende Prozessormikroarchitektur zu bestimmen. Aus diesem Grund wird für diese Hosting-Umgebungen die durchschnittliche minimale und maximale Leistungsaufnahme pro vCPU aller möglichen Prozessormikroarchitekturen verwendet. Die Koeffizienten werden dem Quellcode der Cloud Carbon Footprint Software entnommen und in der Tabelle visualisiert (siehe Tabelle 1, Seite 41). Da für die verwendete Azure VM (siehe 4.) bisher keine Daten für Prozessormikroarchitektur Intel Ice Lake in dem Cloud Carbon Footprint-Datensatz vorliegen, werden diese analog zu der Methodik der Cloud Carbon Footprint-Software (siehe 3.3.3.) berechnet. Die Ergebnisse werden ebenfalls in der folgenden Tabelle gerundet visualisiert (siehe Tabelle 1, Seite 41).

Hosting-Umgebung	W_{min}/vCPU	W_{max}/vCPU
Azure App Service	$0,74$	$3,54$
Azure Container Instance	$0,74$	$3,54$
Azure Function (Verbrauchsplan)	$0,74$	$3,54$
Azure VM	$0,86$	3.92

Tabelle 1: Minimale und maximale Leistungsaufnahme in Watt pro vCPU und Hosting-Umgebung

Bei der Analyse des Quellcodes der Cloud Carbon Footprint-Software fällt auf, dass diese für Services, wie bspw. den Azure App Service, keine Berechnung der Scope 3-Emissionen vornimmt. Dies liegt voraussichtlich daran, dass dafür die Hardwarekonfiguration aufgrund der fehlenden Transparenz der Azure-Cloud nicht bekannt ist.

Es besteht die Annahme, dass Services dieser Art auf verhältnismäßig großen Servern des Typs General-Purpose ausgeführt werden, da diese ein balanciertes Verhältnis aus CPU-Leistung und verfügbarem Arbeitsspeicher besitzen und damit den typischen Workload eines Web-Servers sehr gut bedienen können [43]. Aus diesem Grund wird sich dazu entschieden, die Scope 3-Emissionen für die Hosting-Umgebungen Azure App Service, Azure Container Instance und Azure Function durch die Mittelwertbildung der durch die Cloud Carbon Footprint vorberechneten Scope 3-Emissionen der VMs des Typs General-Purpose abzuschätzen. Aufgrund mangelnder Transparenz seitens Intel bezüglich der zugrundeliegenden Hardware, welche für die definierte Azure VM verwendet wird, wird die Abschätzung der Scope 3-Emissionen auch für die Azure VM verwendet.

Durch die Umstellung der Formel für die Berechnung der Scope 3-Emissionen nach Cloud Carbon Footprint (siehe 3.3.7.), werden die durchschnittlichen Scope 3-Emissionen und deren Standardabweichung in Gramm CO_2-äquivalent pro vCPU-Stunde ($gCO_2e/\text{vCPU}h$) berechnet (siehe Tabelle 2, Seite 42). Dabei wird von einer Serverbetriebsdauer von 4 und 6 Jahren ausgegangen.

Serverbetriebs-dauer	Durchschnitt ($gCO_2e/\text{vCPU}h$)	Standardabweichung ($gCO_2e/\text{vCPU}h$)
4 Jahre	$0,48$	$0,09$
6 Jahre	$0,32$	$0,06$

Tabelle 2: Scope 3-Emissionen pro vCPU-Stunde

Die PUE der Microsoft Azure-Cloud wird ebenfalls der Methodik der Cloud Carbon Footprint-Software entnommen. Diese beträgt durchschnittlich $1,185$ [11].

Für die Berechnung der Scope 2-Emissionen werden die Umrechnungsfaktoren, welche durch die European Environment Agency erfasst und publiziert werden, aus dem Jahr 2020 verwendet (siehe Tabelle 3, Seite 43). [44]

Stromnetzt	Gramm CO_2-äquivalente Emissionen pro Kilowattstunde (gCO_2e/kWh)
Europa (27 Länder, 2020)	229
Schweden (2020)	8
Deutschland (2020)	314

Tabelle 3: Emissionsfaktoren der Stromnetze (Scope 2) im Jahr 2020

8 Messergebnisse

Im Folgenden werden die erfassten Messergebnisse präsentiert und erläutert.

8.1 Latenz der Datenbanken

Zunächst wird die Latenz von Datenbankinteraktionen betrachtet. Diese sind für die Bewertung der Performance und Usability relevant, da die Leistung der Services transitiv abhängig von der Leistung der Datenbank ist. Ziel der Auswertung der gemessenen Latenzen ist die Bewertung der Auswirkung auf die gesamte Ausführungszeit einzelner Anfragen. Diese werden im Folgenden Kapitel betrachtet.

Zur Wiederholung: Szenarien mit temporal entkoppelter Kommunikation (siehe 4.3.) verwenden die Tabellen einer Azure Cosmos DB und Szenarien mit temporal gekoppelter Kommunikation (siehe 4.4.) eine Azure SQL Datenbank. Zudem führt der Produkt-Query-Service (siehe 4.3.) nur lesende Operationen und der Warenkorb-Command-Service (siehe 4.4.) nur schreibende Operationen in der Datenbank aus. Aus diesen beiden Gründen müssen die Latenzen im Folgenden getrennt betrachtet werden.

Die Latenzen der Datenbanken werden durch Application Insights automatisch erfasst. Durch die Kusto Abfragesprache werden über vordefinierte Funktionen die p-Quantile $p-25$, $p-50$ und $p-75$ bestimmt, welche die Box des Box-Plots definieren. Die durchgestrichene Linie im Inneren der Box ist das $p-50$-Quantil also der Median. Für die oberen und unteren Fühler des Box-Plots werden die $p-10$ und $p-90$-Quantile verwendet. Damit weicht die Visualisierung leicht von dem

Standard ab, ist jedoch für die Bewertung ebenso geeignet. Der Mittelwert ist durch die gestrichelte horizontale Linie visualisiert.

Die Boxen des Box-Plots sind nach Hosting-Umgebung bspw. Azure App Service und Kommunikationstechnologie bspw. Web-API oder Azure Service Bus gruppiert. In jeder Gruppe sind die Boxen zudem aufsteigend nach Auslastung sortiert und farblich gruppiert hervorgehoben.

Betrachtet man die Visualisierung der Azure SQL Datenbanklatenzen (siehe Abb. 14, Seite 45) sieht man, dass sich die meisten Latenzen in alles Auslastungsszenarien unterhalb 20 Millisekunden bewegen. Betrachtet man nur die niedrigen Auslastungsszenarien bewegen sich die Latenzen sogar unterhalb der 5 Millisekunden.

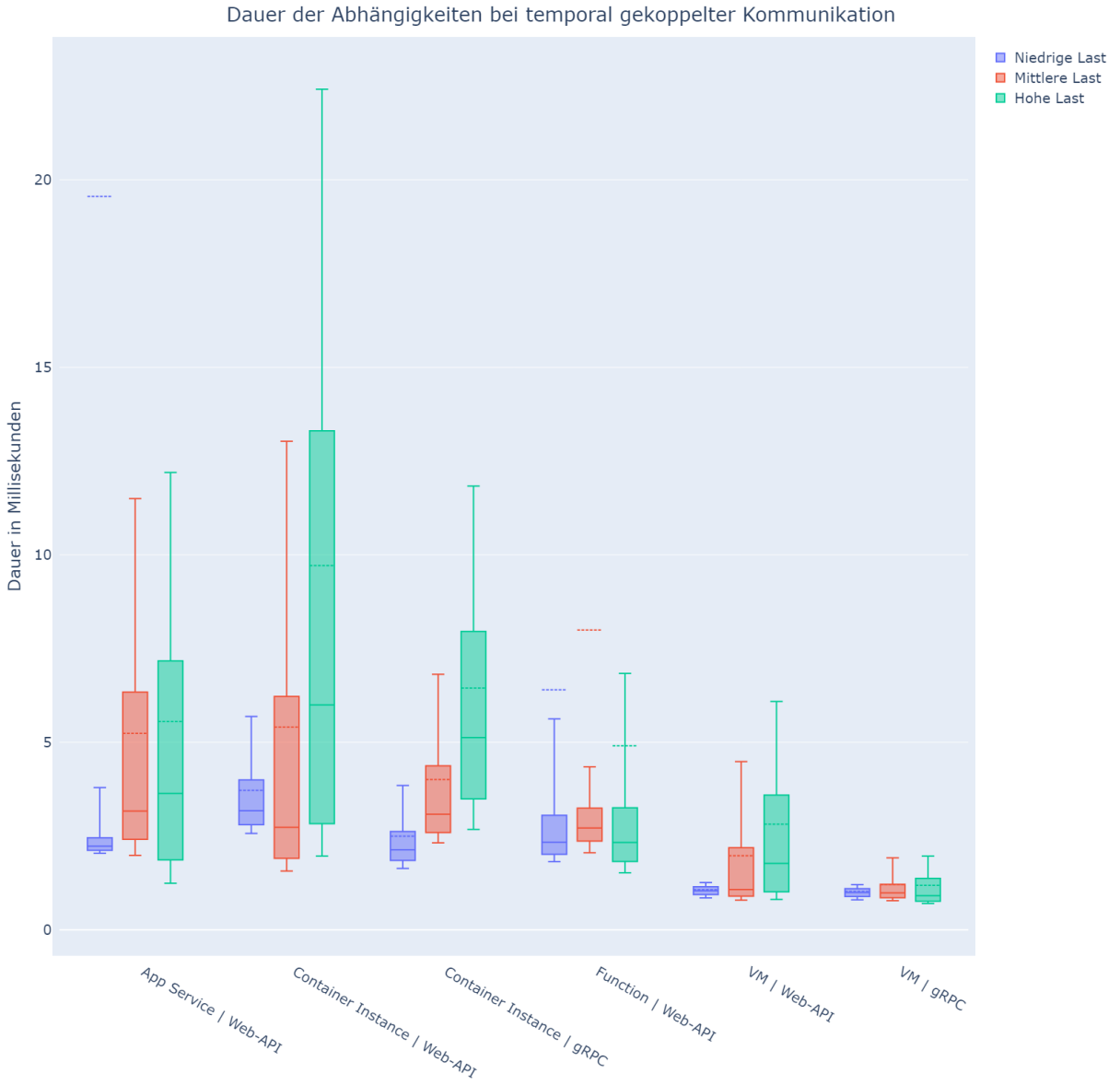

Abbildung 14: Azure SQL Datenbank Latenz

Auffällig ist, dass die Mittelwerte der Latenzen bei der Azure Function-Hosting-Umgebung deutlich oberhalb des Medians liegen und sogar bei niedriger sowie mittlerer Last oberhalb des $p-90$-Quantils liegen.

Des Weiteren ist auffällig, dass beide Szenarien mit der Hosting-Umgebung Azure Container Instances verhältnismäßig schlechter abschneiden als die anderen Hosting-Umgebungen.

Betrachtet man die Latenz der Azure Cosmos DB Tabellen (siehe Abb. 15, Seite 46) fällt beim ersten Blick auf, dass die Latenz bei der Hosting-Umgebung Azure Function deutlich höher ist als alle anderen. Insbesondere bei mittlerer und hoher Last ist der Unterschied enorm.

Bei den Hosting-Umgebungen Azure App Service, Azure Container Instance und Azure VM bewegen sich die Latenzen in allen Auslastungsszenarien in einem unauffälligen Bereich unterhalb der 15 bzw. 10 Millisekunden.

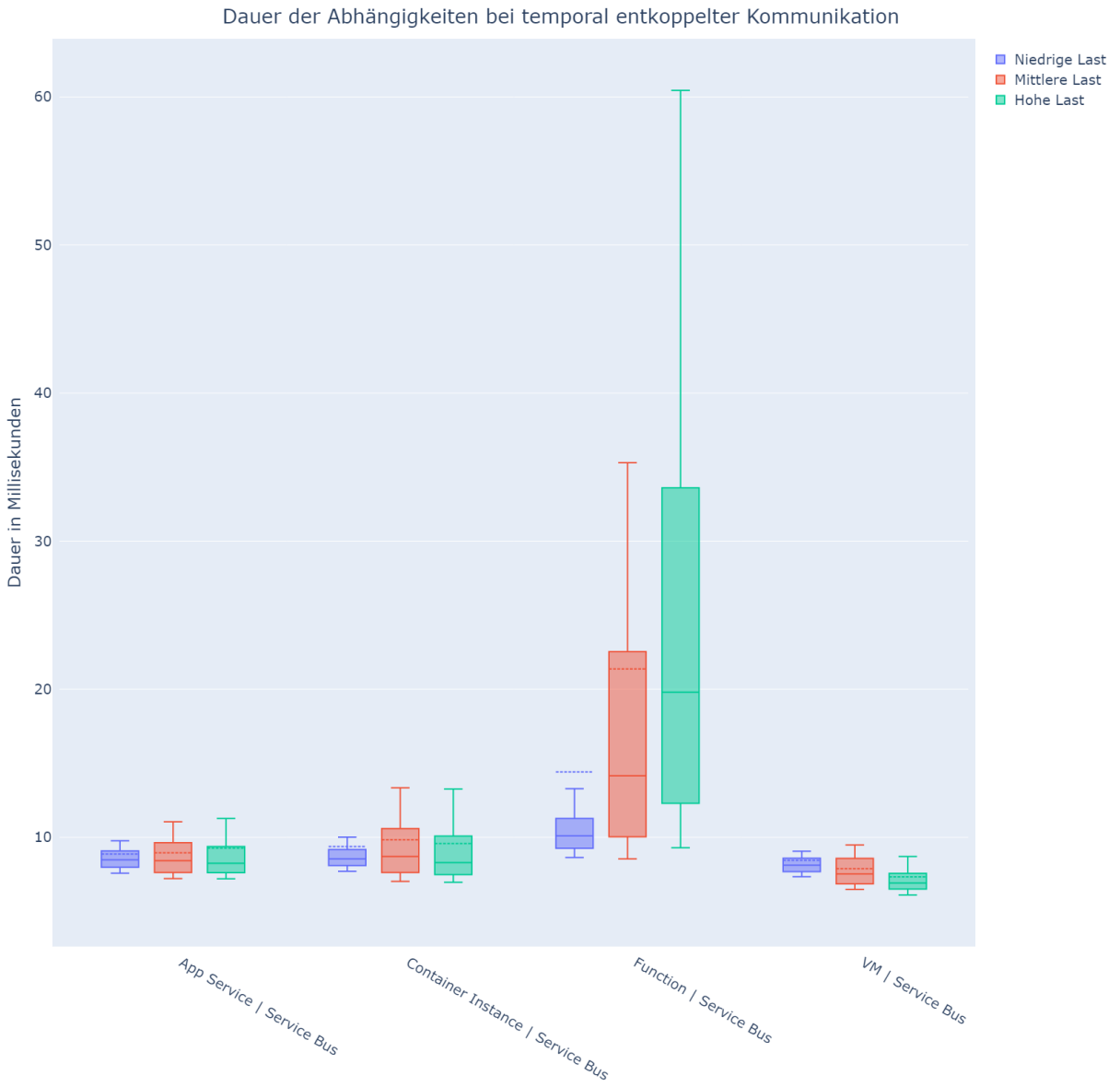

Abbildung 15: Azure Cosmos DB Table Latenz

8.2 Server-seitige Bearbeitungsdauer

Ähnlich wie die Latenz von Datenbankinteraktionen ist die Server-seitige Bearbeitungsdauer relevant für die Bewertung der Performance und Usability der Szenarien. Die Server-seitige Bearbeitungsdauer beeinflusst letztendlich direkt die gesamte Bearbeitungsdauer von Nachrichten bzw. Anfragen. Insbesondere für Azure Functions (Verbhrauchsplan) ist die Metrik zusätzlich ausschlaggebend für die Kostenabschätzung.

Die Server-seitige Bearbeitungsdauer von Anfragen ist eine automatisch durch Application Insights erfasste protokollbasierte Metrik. Sie beschriebt nur die Zeit, die Server-seitig nach dem Eintreffen einer Anfrage bei temporal gekoppelter Kommunikation bzw. nach dem Eintreffen einer Nachricht bei temporal entkoppelter Kommunikation benötigt wird. Sowohl Netzwerklatenz als auch die Zeit, die eine Nachricht in der Queue eines Message-Brokers verbracht hat, ist somit nicht in der Metrik mit inbegriffen.

Die Box-Plots der Server-seitigen Bearbeitungsdauer von Anfragen wird analog zu dem Box-Plot der Datenbanklatenzen gezeichnet. Dies bedeutet insbesondere, dass ebenfalls die $p - 10$ und $p - 90$-Quantil für die oberen und unteren Fühler des Box-Plots verwendet werden. Die Boxen sind ebenfalls gruppiert nach Hosting-Umgebung und Kommunikationstechnologie. Innerhalb der Gruppierung werden die einzelnen Auslastungsszenarien zusätzlich farblich gruppiert und hervorgehoben.

Betrachtet man die Bearbeitungsdauer der Anfragen bei Szenarien mit temporal gekoppelter Kommunikation (siehe Abb. 16, Seite 48) fällt direkt auf, dass die Szenarien mit der Hosting-Umgebung Azure VM wesentlich weniger Zeit für die Bearbeitung von Anfragen benötigen und deutlich konstanter sind als die restlichen Szenarien. Vergleicht man zusätzlich, bei gleicher Hosting-Umgebung Azure VM, die beiden Kommunikationstechnologien Web-API und gRPC fällt außerdem auf, dass gRPC in Bezug auf die Dauer und Streuung deutlich besser abschneidet als die Web-API. Dies gilt insbesondere bei steigender Last. Bei der Hosting-Umgebung Azure Container Instance ist es nicht möglich diese Beobachtung zu pauschalisieren. Genau wie bei den Datenbanklatenzen (siehe Abb. 14, Seite 45) schneiden die Szenarien mit der Hosting-Umgebung Azure Container Instance verhältnismäßig schlechter ab als die anderen Szenarien.

Auffällig ist, dass bei der Hosting-Umgebung Azure Function sowohl bei den Datenbanklatenzen (siehe Abb. 14, Seite 45) als auch bei der Bearbeitungsdauer der Requests die Mittelwerte deutlich oberhalb des $p - 90$-Quantils liegen.

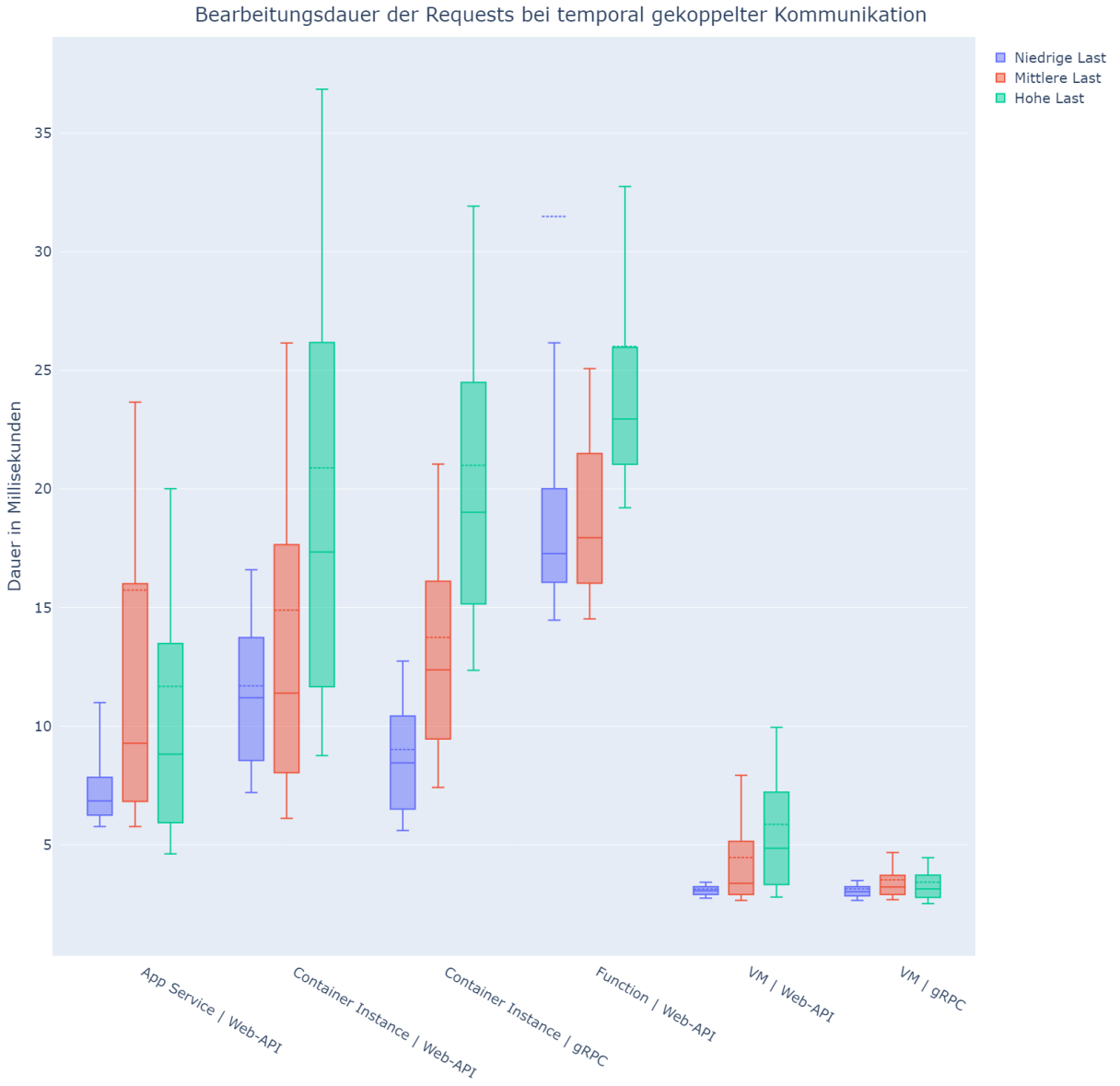

Abbildung 16: Bearbeitungsdauer der Nachrichten bei temporal gekoppelter Kommunikation

Betrachtet man die Server-seitige Bearbeitungsdauer der Nachrichten bei Szenarien mit temporal entkoppelter Kommunikation (siehe Abb. 17, Seite 49) fällt direkt auf, dass das Szenario mit der Hosting-Umgebung Azure Function extrem von den anderen Szenarien abweicht. Ein ähnli-

ches Verhalten lässt sich bereits bei der Latenz der Datenbank in der Hosting-Umgebung Azure Function (siehe Abb. 15, Seite 46) beobachten.

Die Bearbeitungsdauer der Nachrichten in den Szenarien mit der Hosting-Umgebung Azure App Service, Azure Container Instance und Azure VM liegt insgesamt deutlich unterhalb der 20 Millisekunden. Ein ähnliches Verhalten ist bereits bei den Datenbanklatenzen (siehe Abb. 15, Seite 46) der Hosting-Umgebungen zu beobachten. Tendenziell schneidet die Hosting-Umgebung Azure VM insgesamt am besten ab.

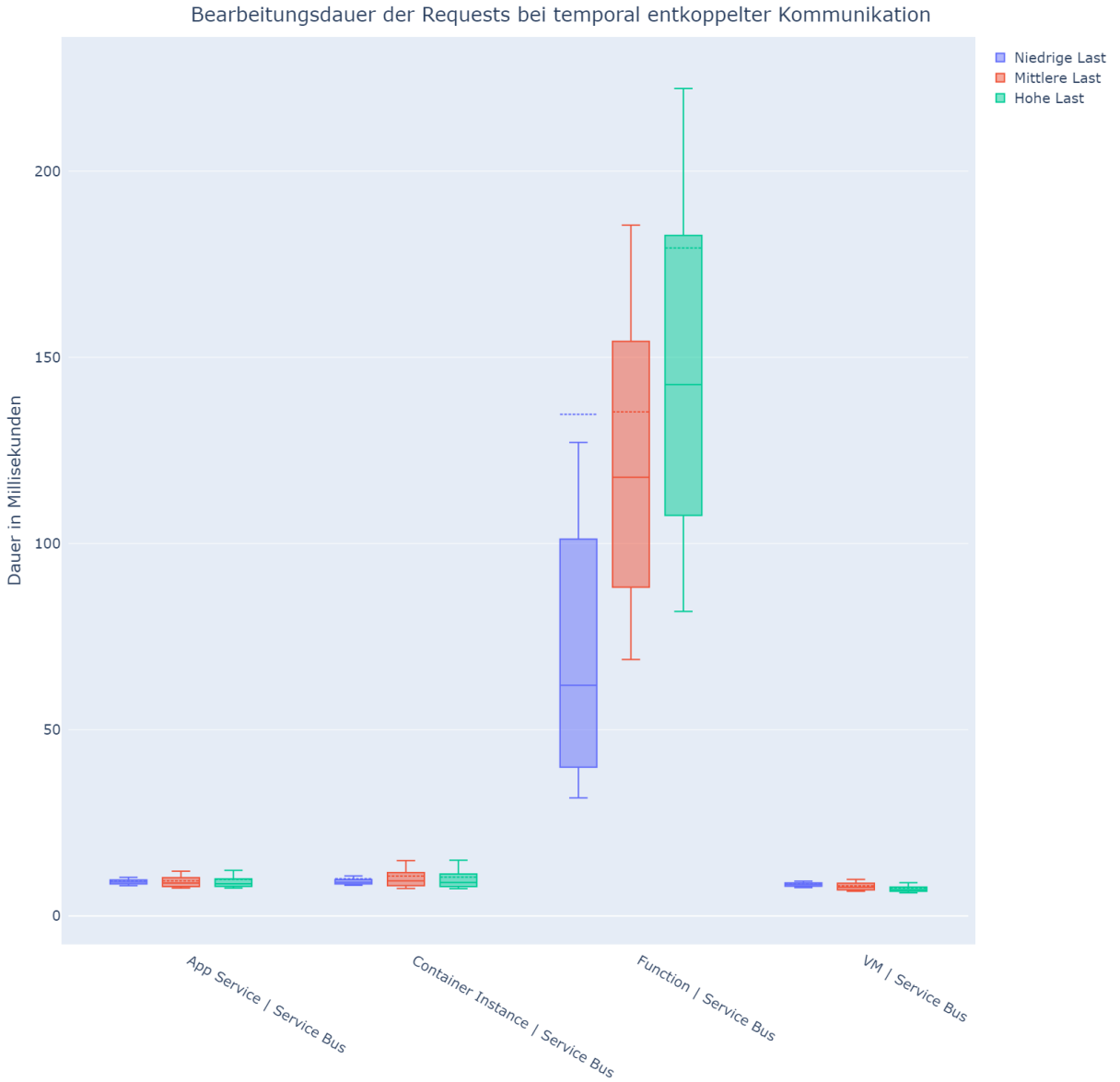

Abbildung 17: Bearbeitungsdauer der Requests bei temporal entkoppelter Kommunikation

8.3 Zuverlässigkeit und Latenz der Anwendung

Im Folgenden werden die Messwerte der gesamten Bearbeitung von Anfragen bzw. Nachrichten betrachtet. Diese können für die Bewertung der Performance und Usability verwendet werden. Im Falle der temporal gekoppelten Kommunikation handelt es sich um die Messwerte welche von der Bibliothek NBomber Client-seitig erfasst werden und im Falle der temporal entkoppelten Kommunikation um die Messwerte, welche durch die eigene protokollbasierte Metrik erfasst werden.

Betrachtet man den Durchsatz und die gesamten Latenzen der Szenarien mit temporal gekoppelter Kommunikation bei niedriger (siehe Tabelle 4, Seite 50), mittlerer (siehe Tabelle 5, Seite 51) und hoher Auslastung (siehe Tabelle 6, Seite 51), fallen vor allem zwei Dinge auf: Zum einen sind die beiden Szenarien mit der Hosting-Umgebung Azure Container Instance bei hoher Last nicht in der Lage den erforderlichen Durchsatz von 250 Anfragen pro Sekunde zu erbringen. Zum anderen ist zu sehen, dass die Azure VM in allen Auslastungsszenarien deutlich schneller als alle anderen Hosting-Umgebungen sind. Insgesamt scheint gRPC etwas performanter und zuverlässiger als eine Web-API zu sein. Mit Ausnahme der Azure Container Instances, liegen die Latenzen insgesamt einem Bereich, in dem das Antwortverhalten als direkte Manipulation wahrgenommen werden kann.

Szenario	Anfragen pro Sekunde	Summe der Anfragen / Nachrichten		p-Quantile der Dauer in Millisekunden			
		Erfolgreich	Nicht erfolgreich	$p-50$	$p-75$	$p-95$	$p-99$
App Service | Web-API	2,5	17821	650	12,96	14,24	19,82	28,42
Container Instance | Web-API	2,5	17976	0	16,61	19,65	25,25	30,94
Container Instance | gRPC	2,5	17976	0	12,36	14,77	20,9	28,98
Function | Web-API	2,5	17976	61	27,98	32,14	53,6	119,74
VM | Web-API	2,5	17976	0	4,36	4,81	5,59	6,56
VM | gRPC	2,5	17976	0	4,37	4,72	5,54	7,11

Tabelle 4: Durchsatz und Latenzen der gesamten Bearbeitungsdauer (temporal gekoppelte Kommunikation, niedrige Last)

Szenario	Anfragen pro Sekunde	Summe der Anfragen / Nachrichten		p-Quantile der Dauer in Millisekunden			
		Erfolgreich	Nicht erfolgreich	$p-50$	$p-75$	$p-95$	$p-99$
App Service | Web-API	25	179742	21	19,06	27,73	38,85	50,78
Container Instance | Web-API	25	179774	2	17,97	27,04	42,3	59,74
Container Instance | gRPC	25	179784	0	17,73	22,22	31,87	43,74
Function | Web-API	24,7	178125	855	26,34	30,51	42,98	61,12
VM | Web-API	25	179784	0	5,55	8,86	12,8	16,59
VM | gRPC	25	179788	0	4,97	5,95	7,77	10,2

Tabelle 5: Durchsatz und Latenzen der gesamten Bearbeitungsdauer (temporal gekoppelte Kommunikation, mittlere Last)

Szenario	Anfragen pro Sekunde	Summe der Anfragen / Nachrichten		p-Quantile der Dauer in Millisekunden			
		Erfolgreich	Nicht erfolgreich	$p-50$	$p-75$	$p-95$	$p-99$
App Service | Web-API	247,1	1779126	62	19,93	27,23	42,27	64,45
Container Instance | Web-API	232,9	1676595	16	36,61	46,72	70,02	94,59
Container Instance | gRPC	130,9	942700	0	28,19	35,46	51,26	67,46
Function | Web-API	242,7	1747693	142	29,66	36,26	58,3	101,31
VM | Web-API	249,6	1797442	0	9,67	11,58	15,78	21,5
VM | gRPC	249,6	1797133	0	5,52	6,32	7,86	12,54

Tabelle 6: Durchsatz und Latenzen der gesamten Bearbeitungsdauer (temporal gekoppelte Kommunikation, hohe Last)

Betrachtet man den Durchsatz und die Latenzen der Szenarien mit temporal entkoppelter Kommunikation bei niedriger (siehe Tabelle 7, Seite 52), mittlerer (siehe Tabelle 8, Seite 52) und hoher Auslastung (siehe Tabelle 9, Seite 52), fällt vor allen Dingen auf, dass die Azure Functions in allen Auslastungsszenarien am unzuverlässigsten in dem Antwortverhalten sind. Insbesondere das p-99 Quantil ist verhältnismäßig deutlich höher als es bei allen anderen Hosting-Umgebungen der Fall ist. Dennoch sind die Azure Functions in allen Auslastungsszenarien in der Lage den erforderlichen Durchsatz an Anfragen pro Sekunde zu erbringen. Insgesamt schneidet die Azure VM in der Zuverlässigkeit und Latenz am besten ab, dicht gefolgt von dem Azure App Service, der Azure Container Instance und zuletzt der Azure Function. Insbesondere bei hoher Auslastung steigt der Median bei allen Hosting-Umgebungen außer der Azure Function bis oberhalb der 200 Millisekunden. Damit ist die Grenze für die Wahrnehmung als direkte Manipulation von 100 Millisekunden überschritten.

Szenario	Anfragen pro Sekunde	Summe der Nachrichten	p-Quantile der Dauer in Millisekunden			
			$p-50$	$p-75$	$p-95$	$p-99$
App Service | Service Bus	2,496944	17978	21,724967	25,553599	44,061443	102,952337
Container Instance | Service Bus	2,497500	17982	24,265165	27,836821	44,922324	97,034718
Function | Service Bus	2,492222	17944	26,364776	31,735579	54,451796	7246,109041
VM | Service Bus	2,496667	17976	18,380359	19,948572	32,241396	92,186397

Tabelle 7: Durchsatz und Latenzen der gesamten Bearbeitungsdauer (temporal entkoppelte Kommunikation, niedrige Last)

Szenario	Anfragen pro Sekunde	Summe der Nachrichten	p-Quantile der Dauer in Millisekunden			
			$p-50$	$p-75$	$p-95$	$p-99$
App Service | Service Bus	24,979167	179850	38,815790	45,300036	62,243242	120,457069
Container Instance | Service Bus	24,982639	179875	44,196643	53,199121	85,595718	136,457289
Function | Service Bus	24,979167	179850	81,184194	100,617473	151,003778	5682,213137
VM | Service Bus	24,979167	179850	30,269335	35,869159	49,893795	92,339026

Tabelle 8: Durchsatz und Latenzen der gesamten Bearbeitungsdauer (temporal entkoppelte Kommunikation, mittlere Last)

Szenario	Anfragen pro Sekunde	Summe der Nachrichten	p-Quantile der Dauer in Millisekunden			
			$p-50$	$p-75$	$p-95$	$p-99$
App Service | Service Bus	248,888889	1792000	229,013752	335,009515	457,651267	540,448553
Container Instance | Service Bus	248,900972	1792087	237,846038	346,747309	466,941657	547,619052
Function | Service Bus	248,888889	1792000	135,085455	174,927016	4526,779720	19743,051246
VM | Service Bus	248,958333	1792500	206,336005	303,285723	422,372401	496,787021

Tabelle 9: Durchsatz und Latenzen der gesamten Bearbeitungsdauer (temporal entkoppelte Kommunikation, hohe Last)

8.4 CO_2-Emissionskritische Messwerte

Im Folgenden werden die für die CO_2-Emissionen relevanten Messwerte der temporal gekoppelten und temporal entkoppelten Szenarien betrachtet. Der Server-seitig erfasste Gesamtdurchsatz an Anfragen bzw. Nachrichten pro Sekunde eines Szenarios wird in die Betrachtung mit aufgenommen, auch wenn dieser nicht direkt relevant für die Berechnung der CO_2-Emissionen ist. Er ist jedoch wichtig für die gesamte Bewertung eines Szenarios.

Die prozentuale vCPU-Auslastung ist eine von Application Insights erfasste protokollbasierte Metrik. Bei der Hosting-Umgebung Azure Function, welche potenziell mehrere Hosting-Instanzen

verwendet, wird der arithmetische Mittelwert der prozentualen vCPU-Auslastung über alle Hosting-Instanzen gebildet. Die Anfragen bzw. Nachrichten pro Sekunde werden durch die Division der Summe der auf dem Server erfassten Anfragen durch die Laufzeit von zwei Stunden berechnet.

Die Grafik (siehe Abb. 18, Seite 54) ist in vier Zeilen und drei Hauptspalten unterteilt. In der oberen Zeile wird der Gesamtdurchsatz in Anfragen pro Sekunde visualisiert. Die drei Hauptspalten unterteilen die Grafik in die unterschiedlichen Auslastungsszenarien, die aufsteigend sortiert sind. Innerhalb eines Last-Szenarios unterteilt die horizontale Achse die Grafik in die einzelnen Hosting-Umgebungen. Diese sind, zum Zweck der besseren Übersichtlichkeit, zusätzlich gruppiert nach Kommunikationstechnologie. In der zweiten Zeile wird die prozentuale vCPU-Auslastung visualisiert. Die dritte Zeile der Grafik visualisiert die Summe der gemessenen vCPU-Stunden über eine Zeitspanne von einer Stunde. In der letzten Zeile der Grafik werden die gemessenen Wattstunden pro Stunde der Ausführung visualisiert. Die Aufteilung der horizontalen Achse ist in allen Zeilen identisch. Die Betrachtung eines Szenarios erfolgt demnach durch die Betrachtung einer gesamten Spalte der Grafik. Für eine bessere Sichtbarkeit ist die Skala der y-Achse pro Last-Szenario an die gemessenen Wertebereiche angepasst.

Betrachtet man die Anfragen pro Sekunde bei niedriger Last (siehe Abb. 18, Seite 54) fällt eine leichte Variation der Balken auf. Dies entsteht durch die bereits beschriebene leichte Ungenauigkeit bei der Datenabfrage. Hinzu kommt, dass durch fehlgeschlagene Anfragen nicht alle Anfragen bzw. Nachrichten an dem Server ankommen, wodurch eine zusätzliche Variation der Ergebnisse entsteht. Der definierte Zieldurchsatz von 2,5 Anfragen pro Sekunde wird jedoch nahezu in allen Szenarien mit niedriger Last erreicht. Bei mittlerer Last erreichen ebenfalls alle Szenarien die definierten 25 Anfragen bzw. Nachrichten pro Sekunde. Bei hoher Last erreichen die meisten Szenarien den definierten Gesamtdurchsatz von 250 Anfragen bzw. Nachrichten pro Sekunde. Ausnahmen sind die Szenarien 5 und 6 als jene mit der Hosting-Umgebung Azure Container Instance und temporal gekoppelter Kommunikation (gRPC und Web-API). Während die Azure Container Instance mit Web-API als Kommunikationstechnologie nur knapp unterhalb des definierten Gesamtdurchsatzes liegt, bricht der Gesamtdurchsatz der Azure Container Instance mit gRPC als Kommunikationstechnologie sehr stark ein. Der Gesamtdurchsatz aller anderen Hosting-Umgebungen bei hoher Last ist weitestgehend unauffällig.

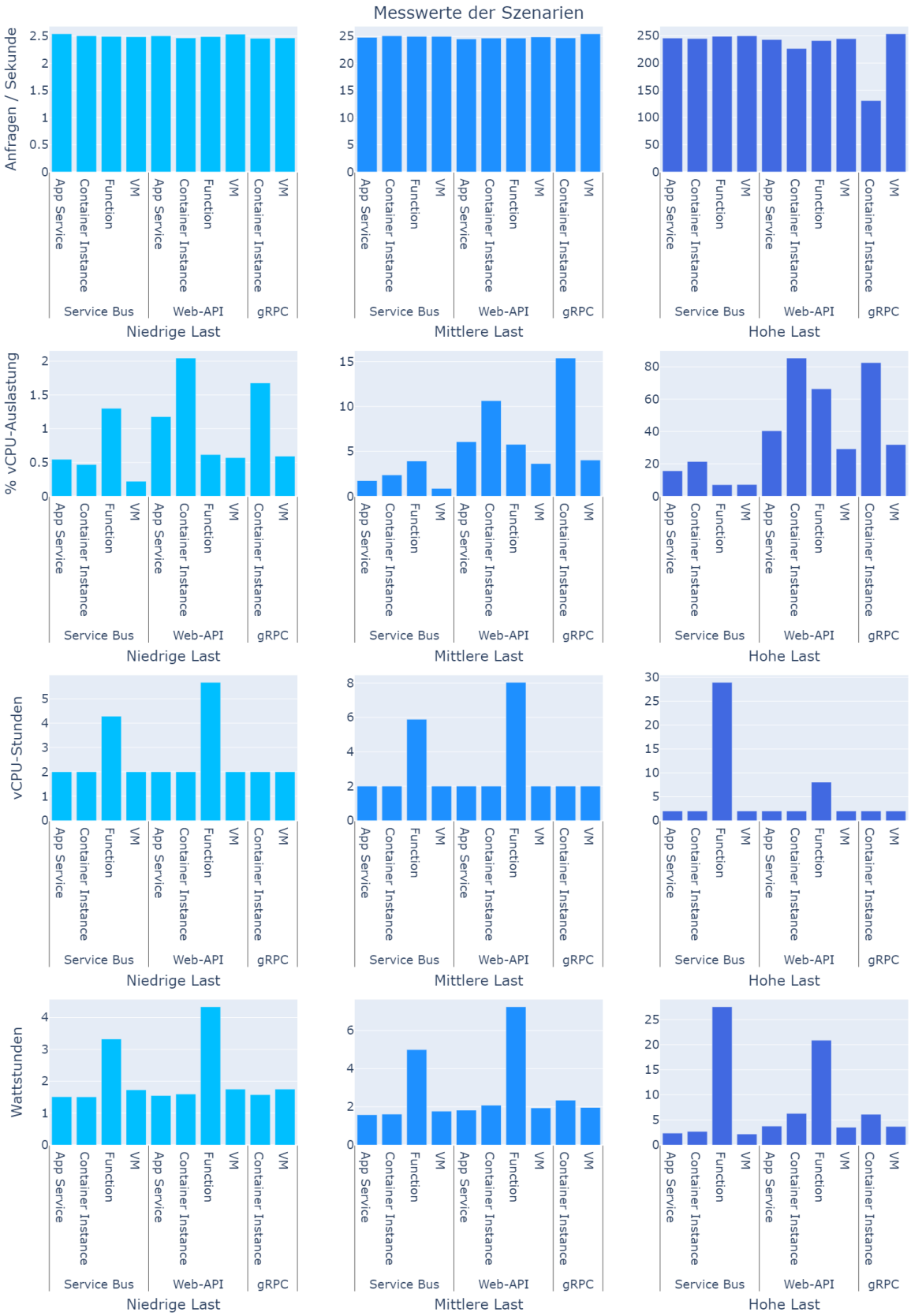

Abbildung 18: Visualisierung der vCPU-Auslastung und Anfragen pro Sekunde

Betrachtet man nun die zweite Zeile der Grafik (siehe Abb. 18, Seite 54), welche die prozentuale vCPU-Auslastung der Szenarien visualisiert, sieht man sehr starke Unterschiede, sowohl bei den Szenarien mit temporal entkoppelter Kommunikation (Azure Service Bus) als auch bei den Szenarien mit temporal gekoppelter Kommunikation (Web-API und gRPC).

Eine interessante Gegebenheit ist, dass die prozentuale vCPU-Auslastung der Hosting-Umgebung Azure Function bei temporal entkoppelter Kommunikation (Azure Service Bus) bei steigender Last im Vergleich zu den anderen Hosting-Umgebungen mit temporal entkoppelter Kommunikation relativ gesehen sinkt. Gleichzeitig steigen jedoch die gemessenen vCPU-Stunden, welche in der dritten Zeile der Grafik visualisiert werden, deutlich an. Dies liegt an dem horizontalen Skalierungsverhalten der Azure Function. Die automatische Skalierung der Azure Function (Azure Service Bus) verteilt demnach bei steigender Last die Anfragen auf sehr viele einzelne Hosting-Instanzen, die wiederum verhältnismäßig wenig vCPU-Auslastung aufweisen.

Betrachtet man nun die Hosting-Umgebung Azure Function bei temporal gekoppelter Kommunikation (Web-API) sieht man deutlich, dass die prozentuale vCPU-Auslastung bei steigender Last verhältnismäßig zu den anderen Hosting-Umgebungen mit temporal gekoppelter Kommunikation ebenfalls steigt. Gleichzeitig steigen auch hier die gemessenen vCPU-Stunden an, jedoch insbesondere bei hoher Last deutlich weniger stark, als es bei der Azure Function mit temporal entkoppelter Kommunikation (Azure Service Bus) der Fall ist. Damit ist hier ein deutlich anderes horizontales Skalierungsverhalten der Azure Function zu beobachten. Die automatische Skalierung der Azure Function verteilt die höhere Last auf verhältnismäßig weniger Hosting-Instanzen, als es bei den Szenarien mit temporal entkoppelter Kommunikation (Azure Service Bus) der Fall ist. Die einzelnen Hosting-Instanzen der Azure Function werden damit insgesamt stärker ausgelastet.

Betrachtet man die dritte Zeile der Grafik (siehe Abb. 18, Seite 54) sieht man, dass die vCPU-Stunden der Hosting-Umgebungen Azure App Service, Azure Container Instance und Azure VM konstant bei 2 vCPU-Stunden pro Stunde Betriebszeit liegen. Dies liegt daran, dass die Szenarien per Definition keine aktivierte automatische Skalierung besitzen und konstant 2 vCPUs verwenden.

Betrachtet man die letzte Zeile der Grafik (siehe Abb. 18, Seite 54) sieht man, dass die gemessenen Wattstunden pro Stunde Betriebszeit bei den Hosting-Umgebungen Azure App Service,

Azure Container Instance und Azure VM bei niedriger und mittlerer Last relativ ähnlich sind. Dies liegt daran, dass die Energieaufnahme dieser Hosting-Umgebungen nur von der zugrundeliegenden Energieaufnahme der Hardware und der prozentualen vCPU-Auslastung abhängt. Wie zuvor beschrieben, sind die vCPU-Stunden pro Stunde Betriebszeit und damit auch die Anzahl der Hosting-Instanzen konstant. Dies zeigt, dass insbesondere bei wenig vCPU-Auslastung und einer konstanten Anzahl an bereitgestellter vCPUs auch die Energieaufnahme verhältnismäßig weniger variiert. Erst bei hoher Last und dementsprechend höherer vCPU-Auslastung machen sich stärkere Unterschiede in der Energieaufnahme bemerkbar.

Vergleicht man die gemessenen vCPU-Stunden der Szenarien mit der Hosting-Umgebung Azure Function und temporal gekoppelter sowie temporal entkoppelter Kommunikation mit den gemessenen Wattstunden, sieht man eine deutliche Korrelation, insbesondere bei niedriger und mittlerer Last. Erst bei einer hohen Last und einer hohen vCPU-Auslastung, wie es bei der Azure Function mit Web-API der Fall ist, sinkt der Einfluss der gemessenen vCPU-Stunden auf die gesamte Energieaufnahme.

8.5 Preisabschätzung

Die Abschätzung der Kosten wird mithilfe des Azure Preisrechners durchgeführt. Dieser erlaubt eine monatliche Kostenabschätzung für Azure Functions, Azure App Services, Azure Container Instances und Azure VMs in der Währung Euro [45].

Im Folgenden wird zur Vereinfachung von 30 Tagen pro Monat ausgegangen. Ebenfalls zur Vereinfachung der Abschätzung werden nur die Kosten für Arbeitsspeicher und vCPU bzw. Rechenzeit in das Gesamtergebnis mitaufgenommen. Da die Preise je nach Land variieren wird die Preisabschätzung zur Vereinfachung einheitlich über die deutschen Tarife durchgeführt.

Für die Preisabschätzung der Azure Functions wird die der verbrauchte Arbeitsspeicher pro ausgeführter Azure Function, die Ausführungsdauer in Millisekunden und die Gesamtanzahl der Ausführungen pro Monat benötigt. Betrachtet man die Verteilung der Server-seitige Bearbeitungsdauer von Anfragen bzw. Nachrichten (siehe 8.2.) würde durch die Verwendung des arithmetischen Mittelwertes der Bearbeitungsdauer schnell zu einer Überabschätzung der Kosten führen. Aus diesem Grund wird der Median der Bearbeitungsdauer für die Preisabschätzung verwendet (siehe Abb. 10,

Seite 57). Bei einer niedrigen Auslastung von $2,5$ Anfragen bzw. Nachrichten pro Sekunde beträgt die Gesamtanzahl der Ausführungen für 30 Tage $6.480.000$, bei mittlerer Auslastung (25 Anfrage pro Sekunde), entsprechend $64.800.000$ und bei hoher Auslastung (250 Anfragen pro Sekunde) $648.000.000$. Für die Arbeitsspeichergröße wird bei der Preisabschätzung der niedrigst mögliche Wert von 128 MB verwendet, da die Berechnungen nicht viel Arbeitsspeicher benötigen.

Szenario	Temporal entkoppelte Kommunikation		Temporal gekoppelte Kommunikation	
	Durchschnitt	Median	Durchschnitt	Median
hohe Auslastung	179.42012	142.73201	26.01091	22.95167
mittlere Auslastung	135.35190	117.79575	146.20300	17.94692
niedrige Auslastung	134.70766	61.92957	31.49006	17.27725

Tabelle 10: Server-seitige Bearbeitungszeit von Anfragen bzw. Nachrichten

Betrachtet man die Ergebnisse fällt auf, dass der Preis bei steigender Auslastung extrem ansteigt (siehe Abb. 11, Seite 57). Bei niedriger Auslastung ist die Preisabschätzung aufgrund des monatlichen Rabattes identisch [46].

Szenario	Temporal entkoppelte Kommunikation	Temporal gekoppelte Kommunikation
hohe Auslastung	304,31 €	150,93 €
mittlere Auslastung	21,41 €	12,65 €
niedrige Auslastung	1,19 €	1,19 €

Tabelle 11: Monatliche Preisabschätzung für die Szenarien mit der Hosting-Umgebung Azure Function

Während die Preise für Azure Functions abhängig von der Auslastung sind, sind die Preise für die Hosting-Umgebungen Azure App Service, Azure Container Instance und Azure VM in Bezug auf Arbeitsspeicher und Rechenzeit bzw. vCPU konstant. Bei dem Einsetzen der Szenariokonfigurationen (siehe 4.) ergeben sich die monatlichen Kosten von $126,63$€ für die Hosting-Umgebung Azure App Service, $95,33$€ für die Hosting-Umgebung Azure Container Instance und $96,75$€ für die Hosting-Umgebung Azure VM.

Betrachtet man die Preisabschätzung im Verhältnis zur Auslastung (siehe Abb. 19, Seite 58), sieht man, dass die Azure Function mit temporal entkoppelter Kommunikation (Azure Service Bus) bis

ca. 80 Anfragen pro Sekunde deutlich günstiger als alle anderen Hosting-Umgebungen ist. Azure Function mit temporal gekoppelter Kommunikation sind sogar bis ca. 160 Anfragen pro Sekunde günstiger als alle anderen Hosting-Umgebungen.

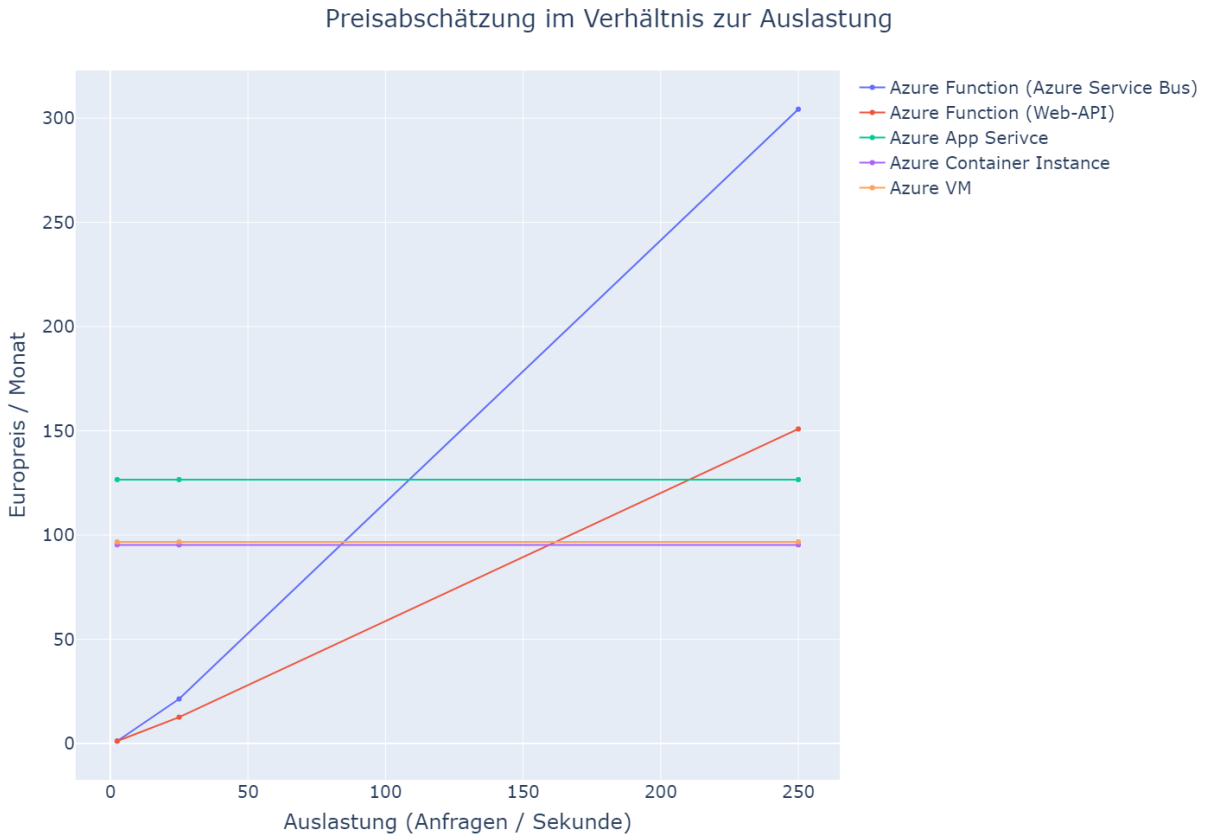

Abbildung 19: Preisabschätzung im Verhältnis zur Auslastung

9 Abschätzung der CO_2-äquivalenten Emissionen

Im Folgenden werden die auf der Basis der Messergebnisse bestimmten CO_2-äquivalenten Emissionen der Szenarien betrachtet, um die CO_2-Intensität der Hosting-Umgebungen und Kommunikationsmuster bzw. Kommunikationstechnologien zu bewerten.

9.1 Auswirkung einer verlängerten Serverbetriebsdauer

Im Folgenden wird betrachtet, welche Auswirkungen die Scope 3 CO_2-Emissionen, die durch die Herstellung eines Serversystems anfallen, in Bezug zur Betriebszeit des Systems stehen. Nicht betrachtet werden dabei die Scope 3 CO_2-Emissionen, die bei dem Bau der gesamten Serverinfrastruktur anfallen, da diese nicht bekannt sind.

Zusätzlich ist zu Beginn zu erwähnen, dass die Abschätzung der Scope 3 CO_2-Emissionen von Serversystemen auf sehr wenigen Daten beruhen und dementsprechend lediglich als grober Richtwert zu betrachten sind.

Der Anteil der Scope 3 CO_2-Emissionen korreliert maßgeblich mit der Betriebszeit eines Servers. Dies liegt daran, dass sich die gebundenen Scope 3 CO_2-Emissionen nicht mehr ändern. Nach der Methodik der Cloud Carbon Footprint-Software wird von einer standardmäßigen Betriebszeit eines einzelnen Serversystem von vier Jahren ausgegangen [20].

Durch die verlängerte Betriebszeit um zwei Jahre ergibt sich, durch die lineare Verteilung der gesamten Scope 3 CO_2-Emissionen auf die gesamte Betriebszeit, eine Einsparung von gerundet $33,332\%$ der gesamten Scope 3 CO_2-Emissionen. Die gesamten Scope 3 CO_2-Emissionen lassen sich demnach um ein Drittel reduzieren, nur durch eine Verlängerung der Serverbetriebszeit um zwei Jahre.

Betrachtet man die CO_2-Emissionen pro Stunde von Szenarien mit temporal gekoppelter Kommunikation, mittlerer Last, einer Serverbetriebszeit von vier Jahren und den Energieemissionsfaktoren des europäischen Stromnetzes (siehe Abb. 20, Seite 60) sieht man deutlich, dass die abgeschätzten Scope 3 CO_2-Emissionen eine wichtige Rolle bei der Betrachtung haben, da sie verhältnismäßig sehr hoch ausfallen. Bei allen Hosting-Umgebungen lässt sich beobachten, dass die Scope 3 CO_2-Emissionen sogar über 50% der gesamten CO_2-Emissionen pro Stunde Ausführungszeit liegen.

Betrachtet man dieselben Szenarien mit einer Serverbetriebszeit von 6 statt 4 Jahren (siehe Abb. 21, Seite 61), also einer Einsparung von einem Drittel der gesamten Scope 3 CO_2-Emissionen ist die Tendenz, dass die Scope 3 und Scope 2 CO_2-Emissionen eher gleichauf liegen.

Scope 3 CO_2-Emissionen spielen somit eine sehr wichtige Rolle bei der Betrachtung aller CO_2-Emissionen. Anteilsmäßig machen sie sogar über 50% der gesamten CO_2-Emissionen aus. Selbst-

verständlich ist dabei zu beachten, dass die Genauigkeit der zugrundeliegenden Daten sehr niedrig ist. Bezieht man jedoch noch die Scope 3 CO_2-Emissionen die durch die gesamte Serverinfrastruktur anfallen mitein, ist davon auszugehen, dass die Dunkelziffer sogar noch größer ausfallen könnte.

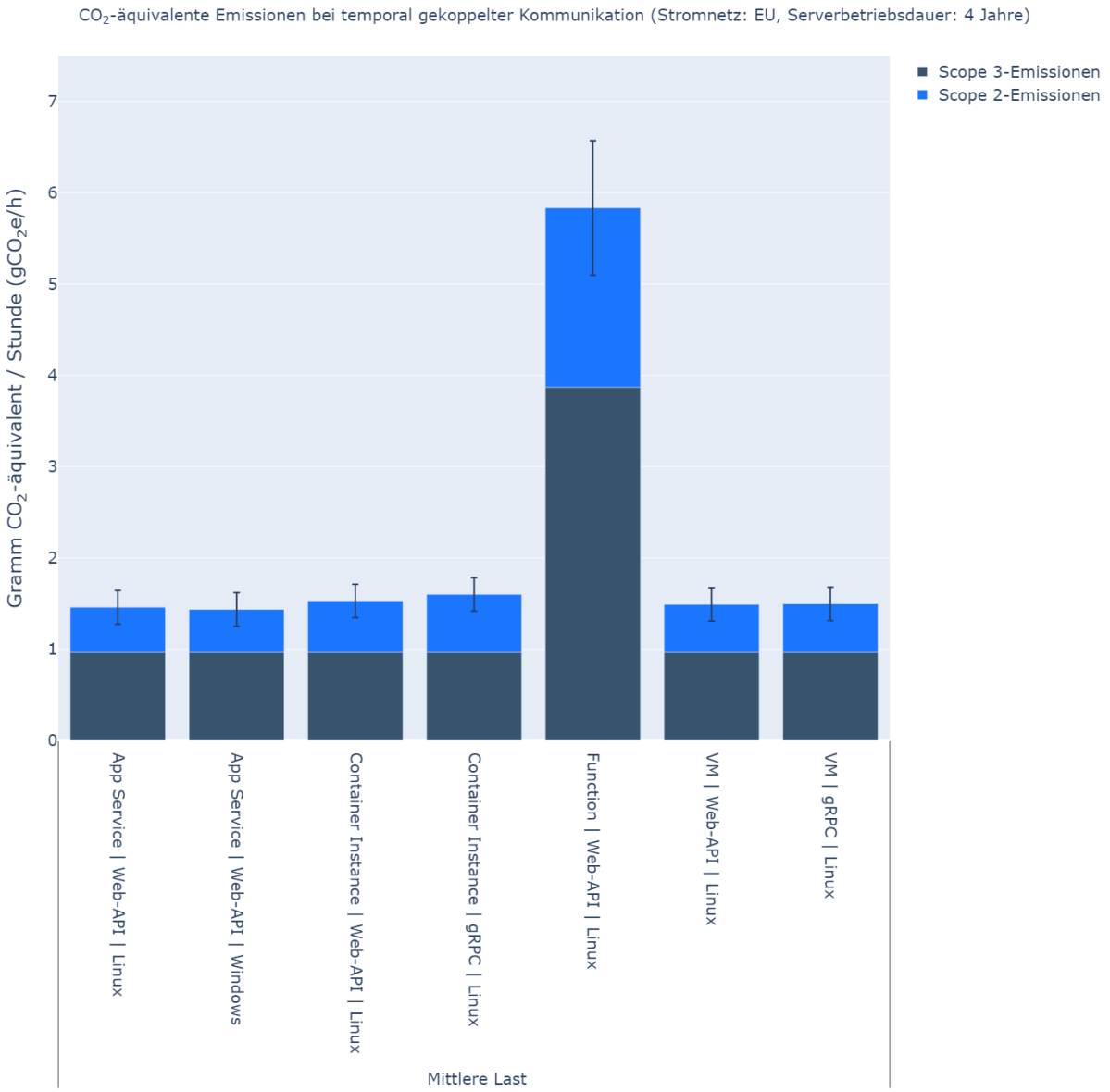

Abbildung 20: CO_2-äquivalente Emissionen bei 4 Jahren Serverbetriebsdauer (Stromnetz: EU, Last: Mittel)

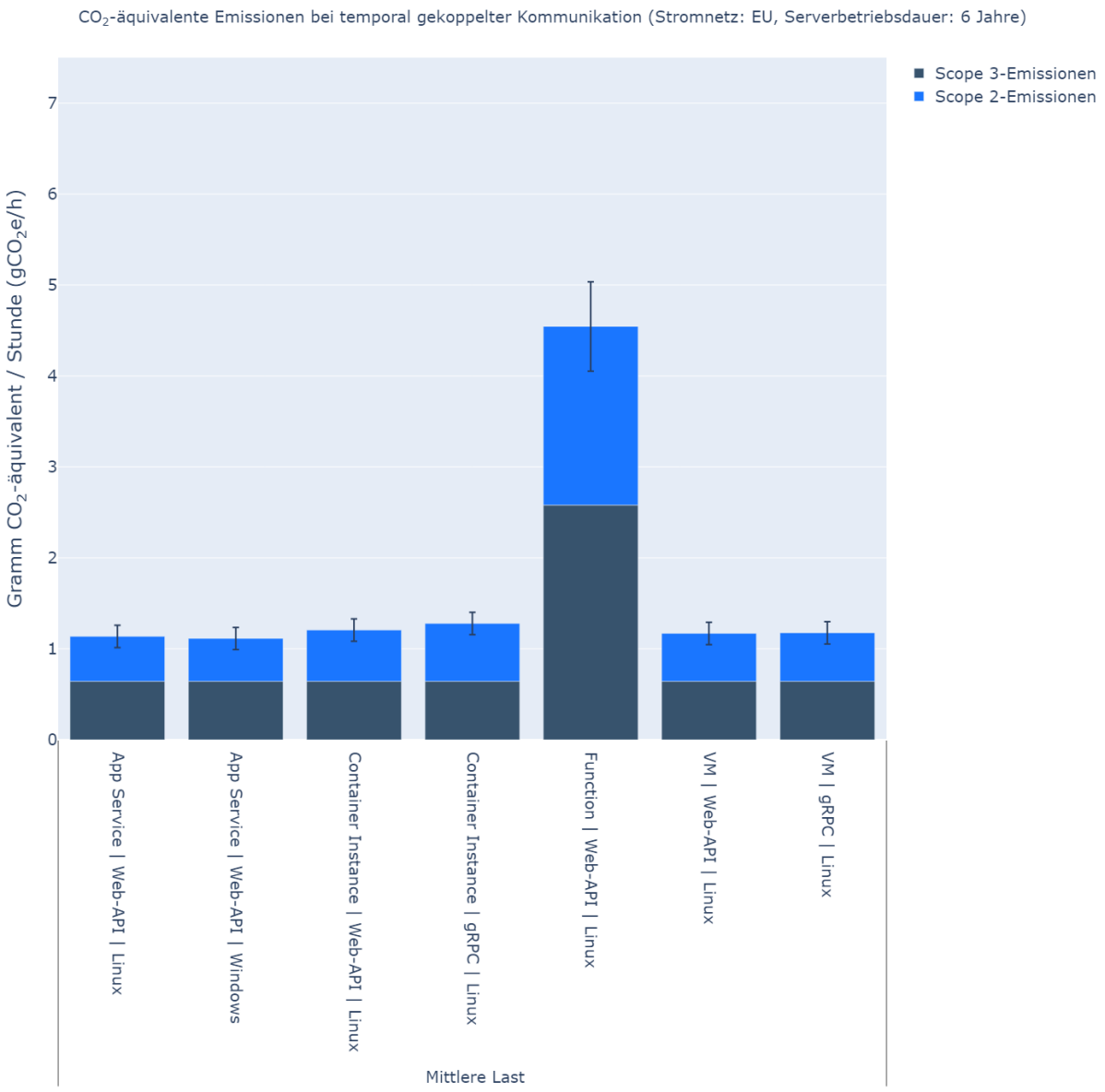

Abbildung 21: CO_2-äquivalente Emissionen bei 6 Jahren Serverbetriebsdauer (Stromnetz: EU, Last: Mittel)

9.2 Auswirkung der CO_2-Intensität des Stromnetzes

Die Auswirkungen der regional unterschiedlichen Scope 2 CO_2-Emissionen spielen bei der Betrachtung der gesamten CO_2-Emissionen eine wichtige Rolle. Um dies zu verdeutlichen, werden im Folgenden die CO_2-Emissionen von Szenarien mit temporal gekoppelter Kommunikation mittlerer Last in unterschiedlichen Stromnetzen betrachtet.

Betrachtet man die CO_2-Emissionen pro Stunde von Szenarien mit temporal gekoppelter Kommunikation, einer Serverbetriebszeit von vier Jahren und den Energieemissionsfaktoren des deutschen Stromnetzes (siehe Abb. 22, Seite 63), sieht man, dass besonders bei einer hohen Last und der damit einhergehenden hohen CPU-Auslastung anteilsmäßig viel Scope 2 CO_2-Emissionen verursacht werden. Insbesondere bei der Hosting-Umgebung Azure Function macht der Scope 2-Anteil zwei Drittel der gesamten CO_2-Emissionen aus. Ein ähnliches Verhalten ist bei den Azure Container Instances zu beobachten. Auch bei dem Azure App Service und bei den Azure VMs ist ein deutlich höherer Anteil der Scope 2 CO_2-Emissionen im Verhältnis zu den Scope 3 CO_2-Emissionen zu beobachten. Bei einer mittleren Last ist tendenziell zu sehen, dass der Scope 2-Anteil und der Scope 3-Anteil eher gleich bzw. der Scope 3-Anteil sogar niedriger als der Scope 2-Anteil ist.

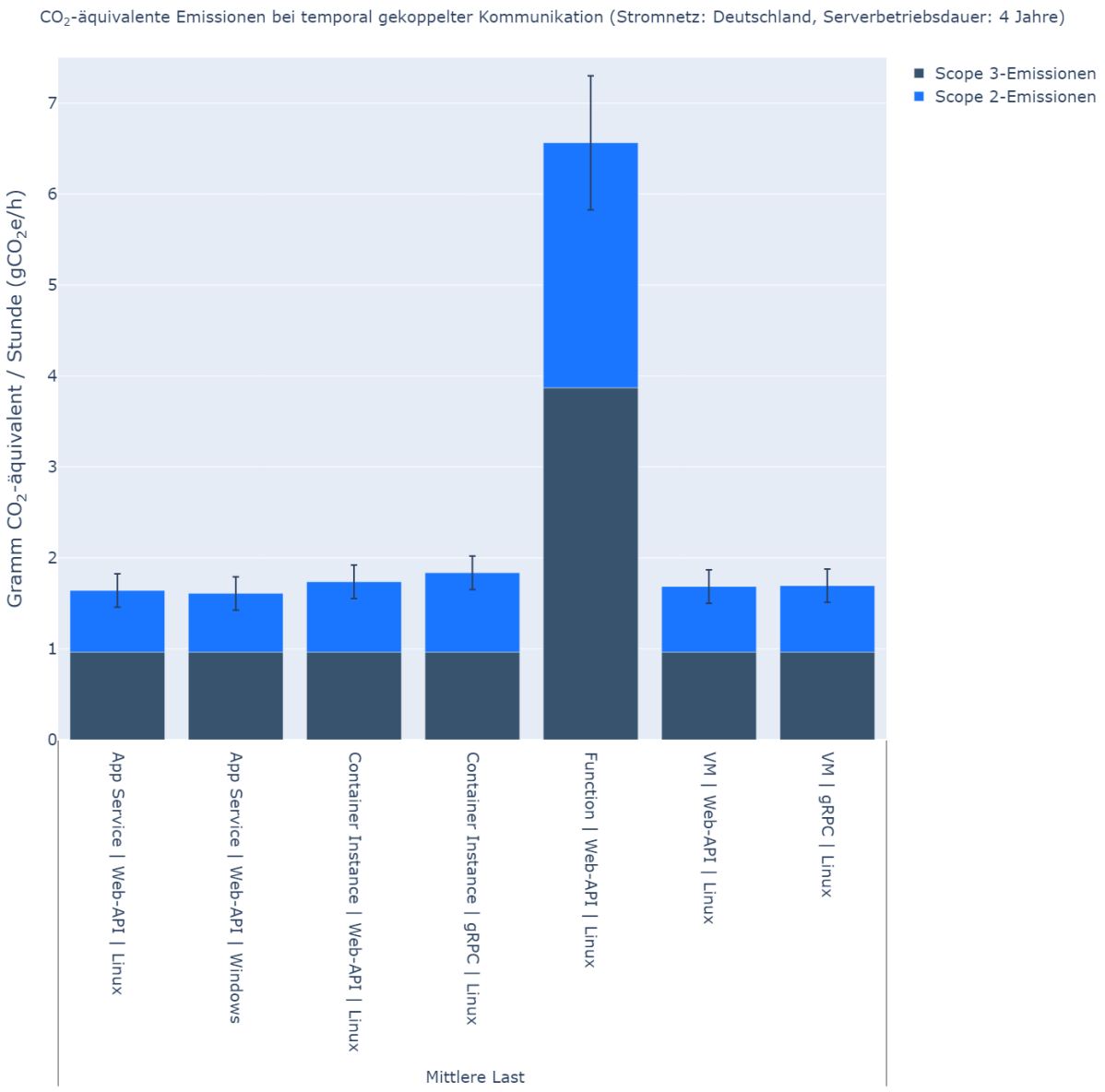

Abbildung 22: CO_2-äquivalente Emissionen in deutschem Stromnetz (Serverbetriebsdauer: 4 Jahre)

Betrachtet man nun die CO_2-Emissionen pro Stunde derselben Szenarien jedoch mit den Umrechnungskoeffizienten des schwedischen Stromnetzes (siehe Abb. 23, Seite 64), sieht man, dass die Scope 2 CO_2-Emissionen lediglich einen Bruchteil der gesamten CO_2-Emissionen ausmachen. Dadurch ist der Unterschied zwischen den einzelnen Last-Szenarien nahezu zu vernachlässigen.

Für den Fall, dass Microsoft bereits auf 100% regenerative Energiequellen setzt, gehen die Scope 2 CO_2-Emissionen nahezu gegen null. Dies hat langfristig zur Folge, dass es darum geht die vorhan-

den Ressourcen bestmöglich vollständig auszulasten um die Scope 3 CO_2-Emissionen der Hardware auf möglichst viele Anwendungen bzw. Kunden verteilen zu können.

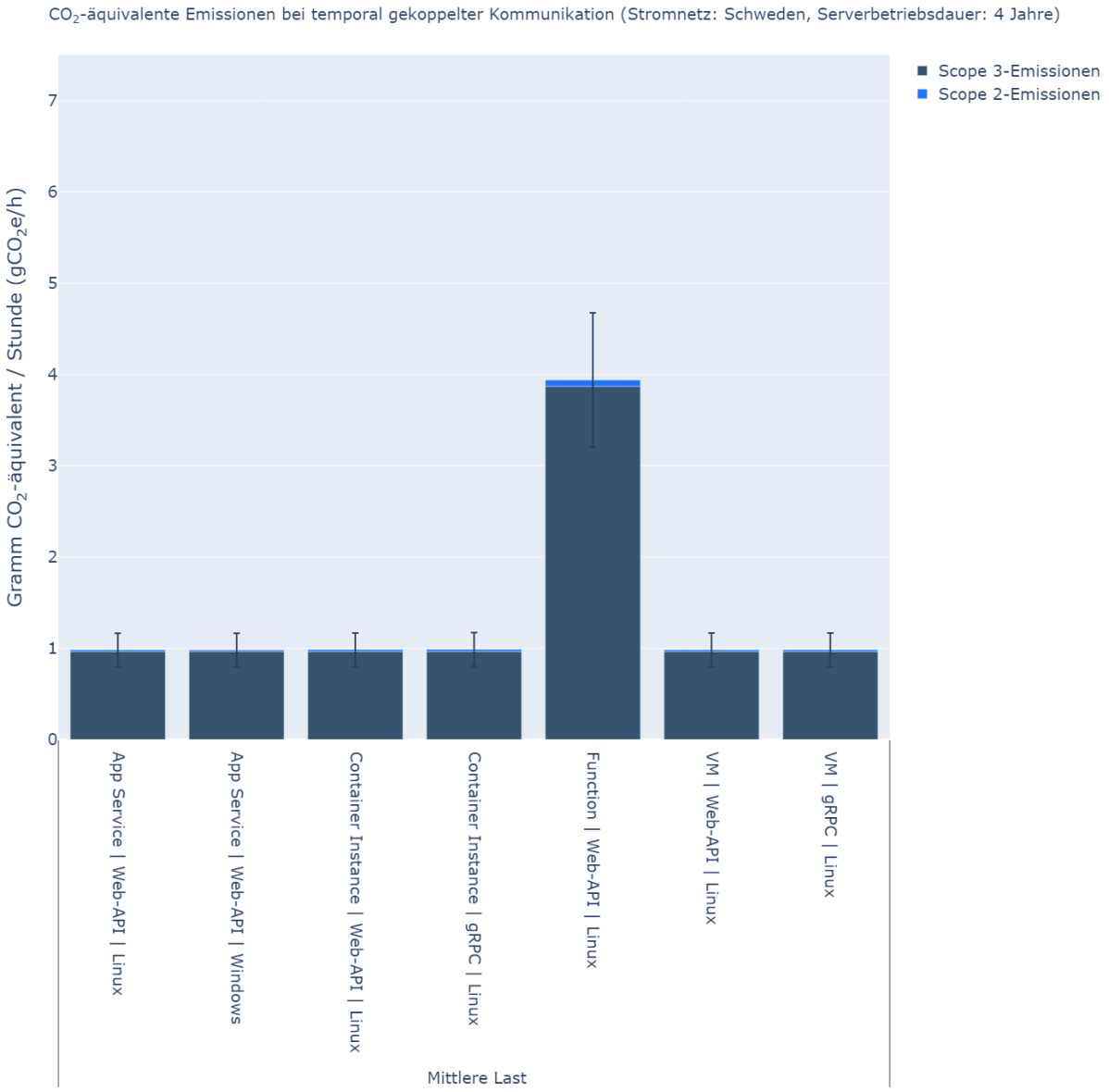

Abbildung 23: CO_2-äquivalente Emissionen in schedischen Stromnetz (Serverbetriebsdauer: 4 Jahre)

9.3 CO_2-Emissionen abhängig von der Hosting-Umgebung und Kommunikationstechnologie

Der zentrale Aspekt der Untersuchung ist es, die Auswirkung der gewählten Bausteine einer Anwendungsarchitektur auf die CO_2-äquivalenten Emissionen zu bestimmen. Dabei bestimmt die gewählte Kommunikationstechnologie eine maßgebliche Rolle bei der Wahl der Hosting-Umgebung. Die Hosting-Umgebungen haben ein unterschiedliches Leistungs- und Skalierungsverhalten, wodurch sich ein unterschiedlicher Energie- bzw. Ressourcenverbrauch (maßgeblich vCPU-Stunden) ergibt (siehe 8.4.). Beobachtungen zeigen, dass die Energieemissionsfaktoren (Scope 2-Emissionen) je nach Stromnetz stark variieren und dementsprechend einen großen Einfluss auf die gesamten CO_2-äquivalenten Emissionen haben (siehe 9.2.). Außerdem zeigt sich, dass durch eine verlängerte Serverbetriebsdauer viel Scope 3-Emissionen eingespart werden können (siehe 9.1.). Aus diesen Gründen ist es notwendig bei der Betrachtung der unterschiedlichen Hosting-Umgebungen und Kommunikationstechnologien, die Energieemissionsfaktoren und Serverbetriebsdauer zu fixieren.

Die Bewertung der Szenarien wird demnach mit den durchschnittlichen europäische Energieemissionsfaktoren und mit den schwedischen Energieemissionsfaktoren durchgeführt. Die schwedischen Energieemissionsfaktoren repräsentieren dabei die Auswirkung eines emissionsarmen Stromnetzes, während die Energieemissionsfaktoren des europäischen Stromnetzes repräsentativ für ein emissionsreiches Stromnetz stehen. Es wird von einer Serverbetriebsdauer von 4 Jahren ausgegangen.

Betrachtet man die Szenarien mit temporal gekoppelter Kommunikation (gRPC und Web-API) im Scope 2 emissionsarmen schwedischen Stromnetz (siehe Abb. 26, Seite 70), sieht man direkt, dass die Hosting-Umgebung Azure Function deutlich schlechter abschneidet als alle anderen Hosting-Umgebungen. Die Scope 3-Emissionen in Gramm pro Stunde sind bereits bei niedriger Last mehr als doppelt so hoch, im Vergleich zu allen anderen Hosting-Umgebungen. Grund dafür ist, wie bereits erwähnt, die automatische Skalierung der Azure Function. Diese startet bereits bei wenig Last mehrere Hosting-Instanzen, wodurch diese deutlich mehr vCPU-Stunden pro Stunde Ausführungszeit generieren (siehe Abb. 18, Seite 54). Bei mittlerer und hoher Last erhöht sich der Scope 3-Anteil der CO_2-Emissionen, bleibt jedoch in beiden Szenarien sehr ähnlich. Bei ho-

her Last steigen die Scope 2-Emissionen der Azure Function noch einmal deutlich im Vergleich zur Azure Function bei mittlerer Last. Durch die niedrigen Energieemissionsfaktoren des schwedischen Stromnetzes haben die durch den Energieverbrauch verursachten Scope 2-Emissionen einen sehr geringen Einfluss auf das Gesamtergebnis.

Bei den Szenarien mit der Hosting-Umgebung Azure App Service, Azure Container Instance und Azure VM ist bei niedriger und mittlerer Last nahezu kein Unterschied der CO_2-Emissionen zu erkennen. Innerhalb dieser Last-Szenarien schwankt die Energieaufnahme zwar leicht (siehe Abb. 18, Seite 54), allerdings gleicht auch hier das emissionsarme Stromnetz die Schwankungen weitestgehend aus. Bei hoher Last sieht man, dass die beiden Azure Container Instances etwas mehr Scope 2-Emissionen als der Azure App Service bzw. die Azure VM verursachen. Wie bereits erwähnt sind die beiden Azure Container Instances bei hoher Last allerdings überlastet und sind demnach ungeeignet (siehe 8.4.).

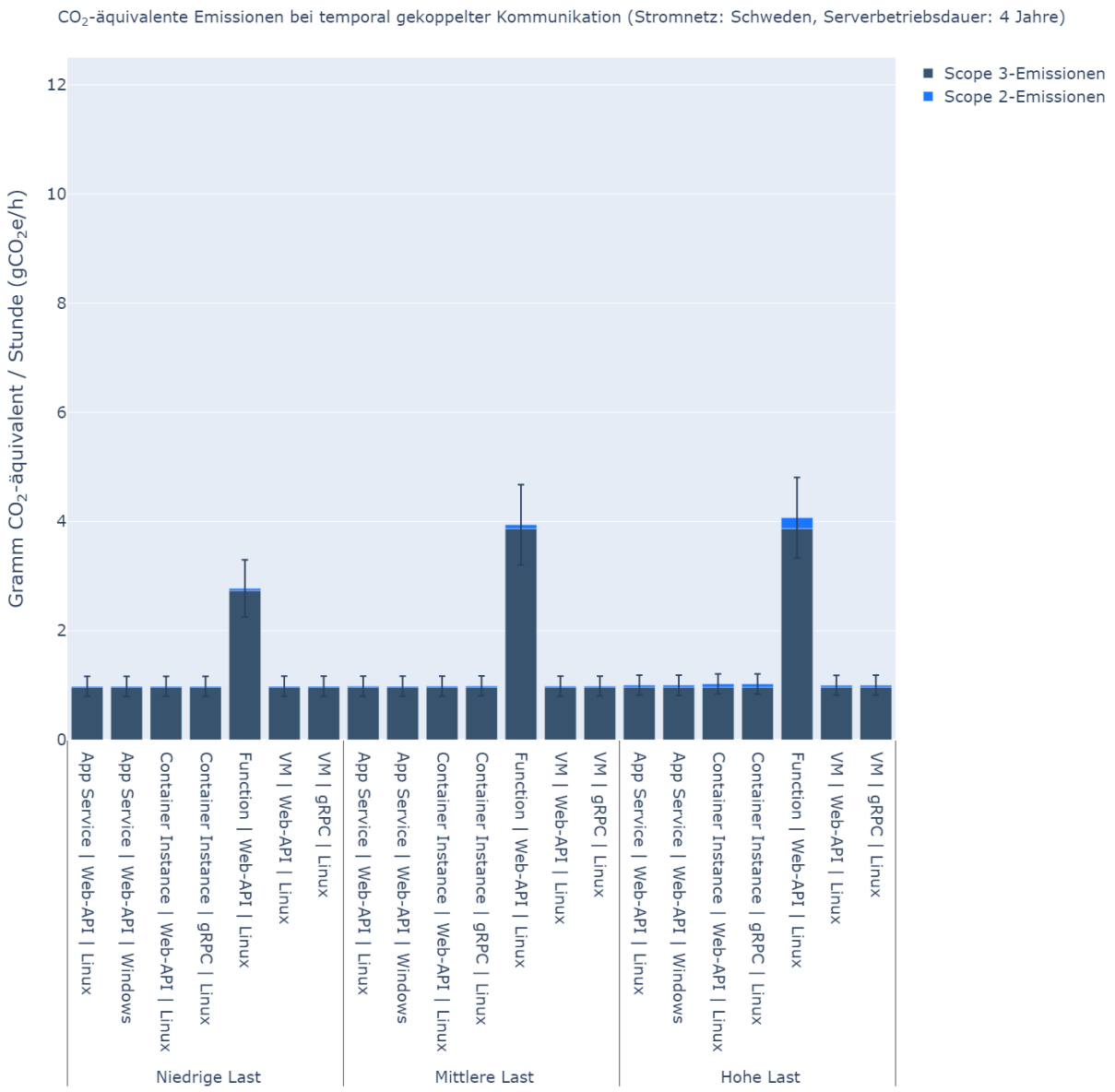

Abbildung 24: CO_2-äquivalente Emissionen bei temporal gekoppelter Kommunikation (Stromnetz: Schweden)

Betrachtet man die Szenarien mit temporal entkoppelter Kommunikation mit den schwedischen Energieemissionsfaktoren, sieht man ebenfalls, dass die Azure Function in allen Auslastungsszenarien deutlich höhere CO_2-Emissionen aufweist. Die CO_2-Emissionen setzen sich größtenteils aus Scope 3-Emissionen zusammen. Betrachtet man zusätzlich den Energieverbrauch der Azure Function mit temporal entkoppelter Kommunikation (Azure Service Bus) und vergleicht diesen mit den anderen Hosting-Umgebungen, sieht man, dass der Energieverbrauch ebenfalls in allen Auslastungsszenarien deutlich über den anderen Hosting-Umgebungen liegt (siehe 8.4.).

Zwischen den Hosting-Umgebungen Azure App Service, Azure Container Instance und Azure VM ist nahezu kein Unterschied der CO_2-Emissionen erkennbar.

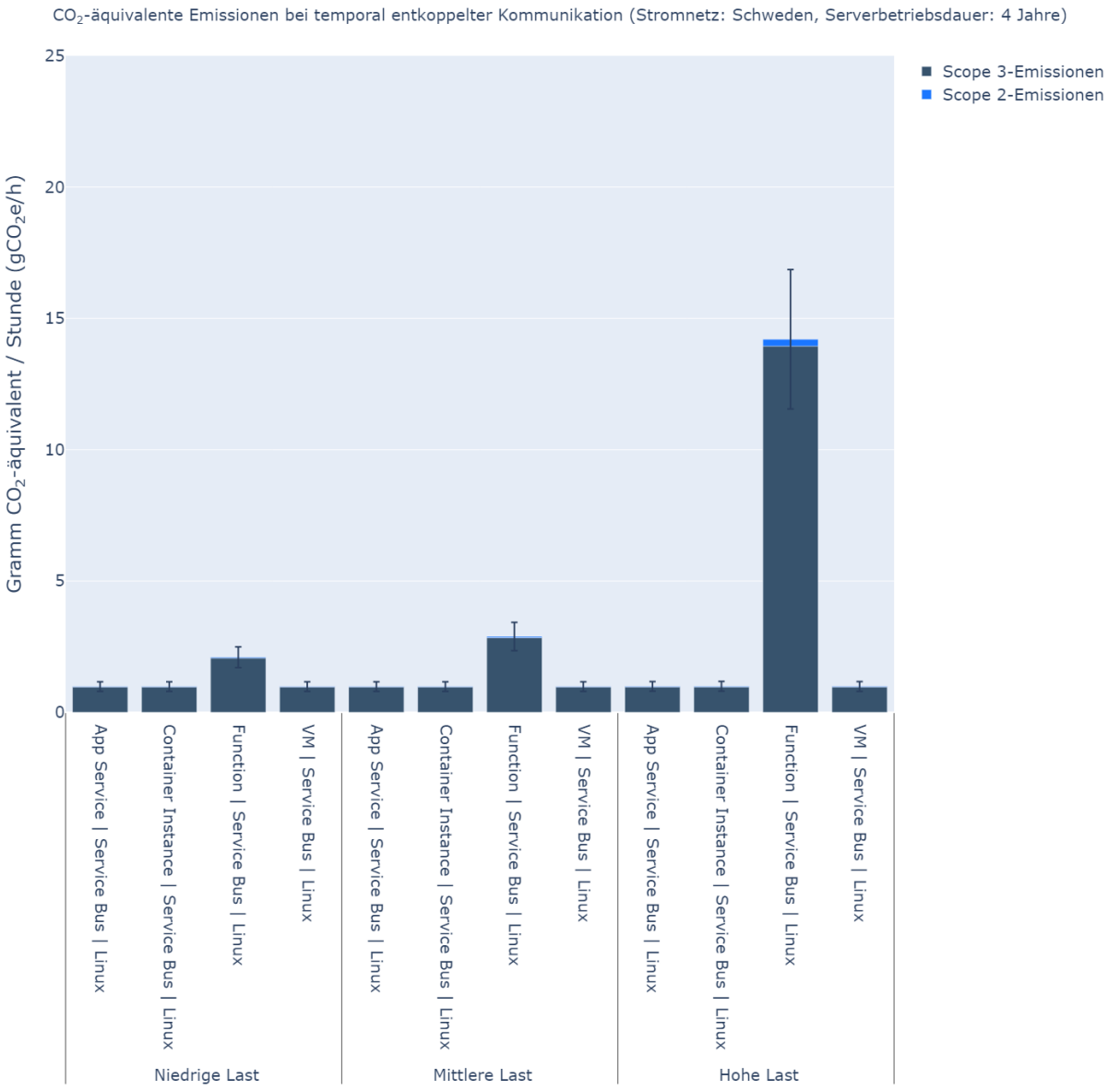

Abbildung 25: CO_2-äquivalente Emissionen bei temporal entkoppelter Kommunikation (Stromnetz: Schweden)

Betrachtet man dieselben Szenarien mit den durchschnittlichen Energieemissionsfaktoren des europäischen Stromnetzes ergibt sich ein völlig anderes Bild. Die Scope 2-Emissionen haben, wie bereits beschrieben, einen deutlich größeren Effekt auf das Gesamtergebnis (siehe 9.2.).

Betrachtet man die Szenarien mit temporal gekoppelter Kommunikation (siehe Abb. 26, Seite 70), fällt auf, dass wie zuvor die Hosting-Umgebung Azure Function verhältnismäßig am schlechtesten abschneidet. In allen Auslastungsszenarien sind die CO_2-Emissionen der Azure Function mindestens doppelt so hoch, wie die der anderen Hosting-Umgebungen. Besonders auffällig bei der Azure Function ist außerdem, dass zwischen mittlerer Last und hoher Last fast ausschließlich die Scope 2-Emissionen zunehmen. Wie bereits beschrieben, kommt dies daher, dass die automatische Skalierung der Azure Function bei hoher Last ca. gleich viele Hosting-Instanzen verwendet, wie es bei der mittleren Last der Fall ist (siehe 8.4.). Diese werden dafür insgesamt stärker ausgelastet und benötigen dementsprechend mehr Energie. Bei niedriger Last lässt sich zwischen den Hosting-Umgebungen Azure App Service, Azure Container Instance und Azure VM sowie zwischen Web-API und gRPC nahezu kein Unterschied der CO_2-Emissionen erkennen. Bei mittlerer Last ist jedoch erkennbar, dass ein kleiner Unterschied der CO_2-Emissionen zwischen Web-API und gRPC innerhalb der Hosting-Umgebung Azure Container Instance besteht. Die Kommunikationstechnologie gRPC verursacht etwas mehr CO_2-Emissionen bei gleicher Auslastung. Außerdem ist erkennbar, dass die Azure Container Instanzen etwas mehr CO_2-Emissionen verursachen als die Hosting-Umgebungen Azure App Service und Azure VM. Azure App Service und Azure VM verursachen bei mittlerer und hoher Last ca. gleich viel CO_2-Emissionen. Allerdings ist auch bei hoher Auslastung in der Hosting-Umgebung Azure VM zu beobachten, dass gRPC tendenziell etwas mehr CO_2-Emissionen verursacht als die klassische Web-API. Der Unterschied ist jedoch aufgrund des großen Fehlerbalkens zu vernachlässigen.

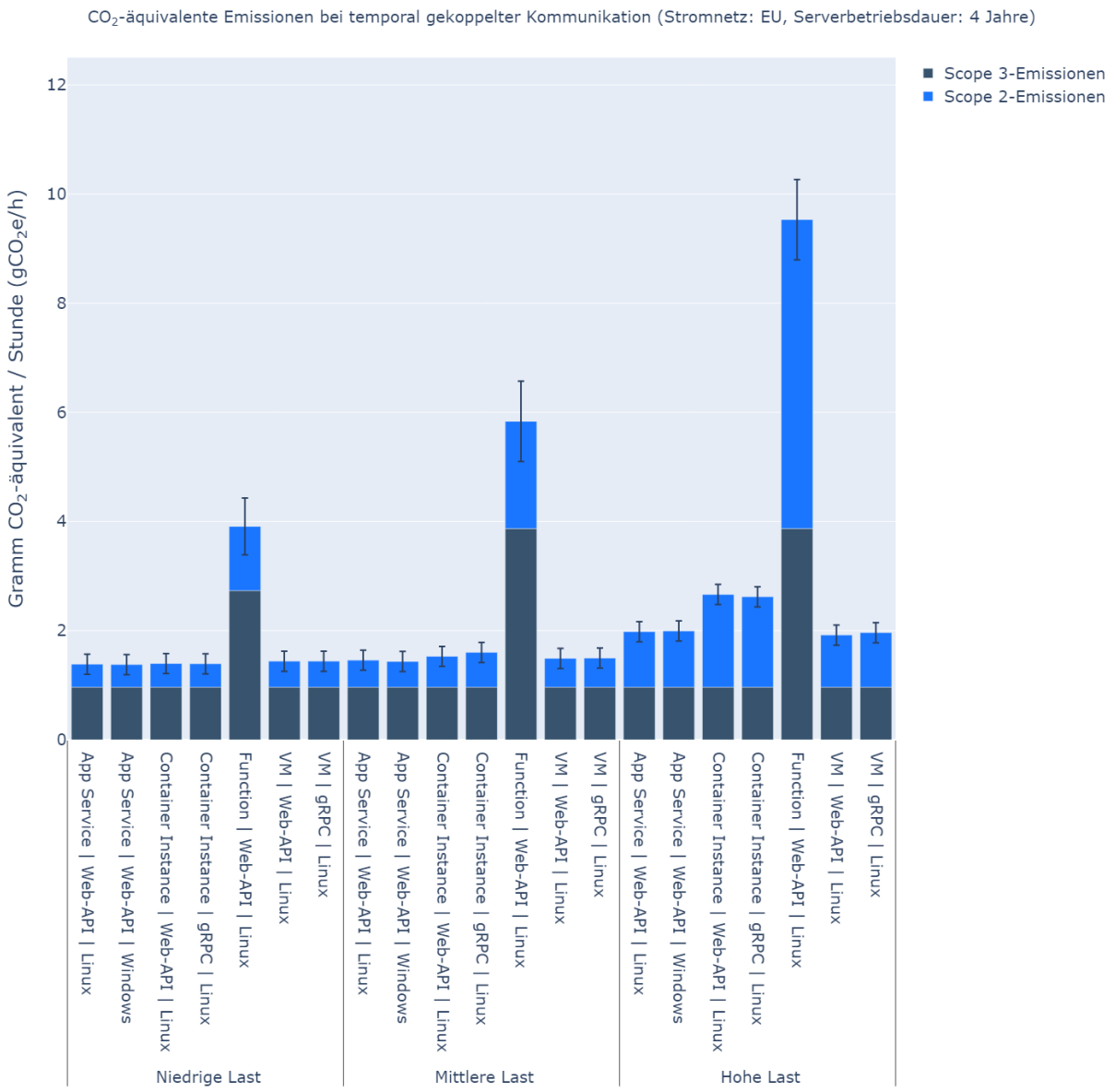

Abbildung 26: CO_2-äquivalente Emissionen bei temporal gekoppelter Kommunikation (Stromnetz: Europa)

Auch bei der Betrachtung der Szenarien mit temporal entkoppelter Kommunikation in Kombination mit den durchschnittlichen Energieemissionsfaktoren des europäischen Stromnetzes (siehe Abb. 27, Seite 71), ist deutlich erkennbar, dass die Hosting-Umgebung Azure Function in allen Auslastungsszenarien am schlechtesten abschneidet. Besonders auffällig ist, dass die Azure Function bei hoher Auslastung ein vielfaches mehr an Scope 3-Emissionen verursacht, als es bei mittlerer bzw. niedriger Auslastung der Fall ist. Dies kommt daher, dass bereits die vCPU-Stunden dieser Azure Function bei hoher Auslastung sehr hoch sind (siehe Abb. 18, Seite 54). Die Ursache dafür

ist die automatische Skalierung der Azure Function (siehe 8.4.). Zwischen den anderen Hosting-Umgebungen ist in allen Auslastungsszenarien nahezu kein wesentlicher Unterschied zwischen den CO_2-Emissionen zu erkennen.

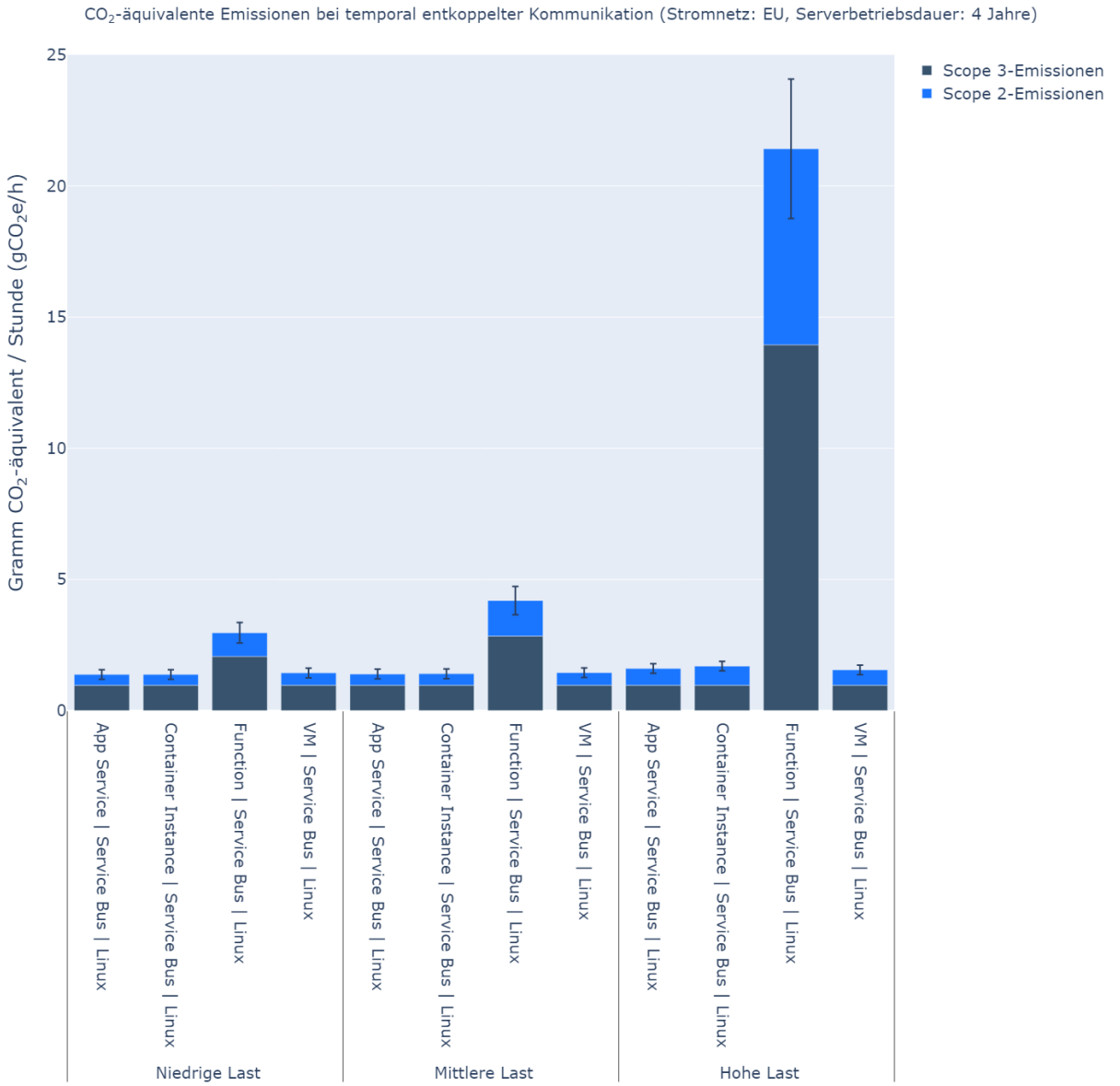

Abbildung 27: CO_2-äquivalente Emissionen bei temporal entkoppelter Kommunikation (Stromnetz: Europa)

10 Diskussion

10.1 Zusammenfassung der Ergebnisse

Die Datenbanklatenzen der Szenarien, welche eine Azure SQL Datenbank verwenden (Szenarien mit temporal gekoppelter Kommunikation), sind insgesamt sehr ähnlich. Das $p-90$-Quantil bei hoher Last übersteigt nur in einem Szenario die 20 Millisekunden. Bei allen anderen Szenarien liegt das $p-90$-Quantil deutlich unterhalb der 15 Millisekunden. Bei den Szenarien, die eine Azure Cosmos DB verwenden (Szenarien mit temporal entkoppelter Kommunikation) ist ein ähnliches Verhalten zu beobachten. Lediglich bei der Hosting-Umgebung Azure Function liegt das $p-90$-Quantil bei hoher Last knapp oberhalb der 60 Millisekunden und damit deutlich höher als alle anderen Szenarien.

Bei der Server-seitigen Bearbeitungsdauer von Anfragen bzw. Nachrichten zeichnet sich ein ähnliches Bild ab. Die Bearbeitungsdauer der Anfragen bei Szenarien mit temporal gekoppelter Kommunikation bewegt sich größtenteils unterhalb der 35 Millisekunden. Die Hosting-Umgebung Azure VM schneidet insgesamt am besten ab. Bei den Szenarien mit temporal entkoppelter Kommunikation (Azure Service Bus) fällt auf, dass die Bearbeitungsdauer der Nachrichten bei der Hosting-Umgebung Azure Function insgesamt deutlich höher ist als bei allen anderen Hosting-Umgebungen. Bei hoher Last liegt dort die Bearbeitungsdauer der meisten Nachrichten zwischen 75 und 250 Millisekunden, während sie bei den anderen Hosting-Umgebungen unterhalb der 25 Millisekunden liegt. Die Abrechnung der Azure Function mit Verbrauchsplan erfolgt nach der Ausführungsdauer. Dementsprechend wirkt sich die Server-seitige Bearbeitungsdauer von Nachrichten bzw. Anfragen auf die Azure Function Preisentwicklung aus. Zusätzlich steht die Server-seitige Bearbeitungsdauer in direktem Zusammenhang mit der gesamten Bearbeitungsdauer von Nachrichten bzw. Anfragen.

Bei der Betrachtung der gesamten Bearbeitungsdauer von Nachrichten bzw. Anfragen, zeigt sich, dass Azure Functions bei der Betrachtung der Zuverlässigkeit und Latenz tendenziell am schlechtesten abschneiden. Außerdem ist zu sehen, dass die Azure Container Instances bei hoher Auslastung und temporal gekoppelter Kommunikation nicht in der Lage sind, die erforderliche Leistung zu erbringen. Die beste Performance und Zuverlässigkeit kann die Hosting-Umgebung Azure VM

erbringen. Dazwischen liegt der Azure App Service und je nach Auslastungsszenario die Hosting-Umgebung Azure Container Instance. Insgesamt schneidet die Kommunikationstechnologie gRPC etwas besser als eine klassische Web-API ab.

Die Hosting-Umgebung Azure Function schneidet sowohl bei temporal gekoppelter als auch bei temporal entkoppelter Kommunikation bei den produzierten vCPU-Stunden und Wattstunden über eine Betriebsdauer von einer Stunde verhältnismäßig schlecht ab. Insbesondere bei hoher Auslastung sorgt die automatische Skalierung der Azure Function bei temporal entkoppelter Kommunikation dafür, dass sehr viele Hosting-Instanzen gleichzeitig gestartet sind. Dies treibt sowohl vCPU-Stunden als auch den Energieverbrauch verhältnismäßig sehr stark nach oben (siehe 8.4.).

Bei der Betrachtung der CO_2-äquivalenten Emissionen (siehe 9.3.) zeigt sich ebenfalls, dass die Azure Functions verhältnismäßig deutlich mehr CO_2-Emissionen verursachen als alle anderen Szenarien. Je nach Wahl des Stromnetzes und den damit einhergehenden unterschiedlichen Energieemissionsfaktoren variieren die CO_2-äquivalenten Emissionen der Hosting-Umgebungen Azure App Service, Azure Container Instance und Azure VM leicht. Eine starke CPU-Auslastung und der damit einhergehende hohe Energieverbrauch spielt jedoch nur bei hohen Energieemissionsfaktoren eine Rolle. Zwischen den unterschiedlichen Kommunikationstechnologien ist nahezu kein Unterschied zu beobachten.

Die Betrachtung der Kostenabschätzung zeigt, dass Azure Functions bis hin zu einer Auslastung von ca. 80 bzw. 160 Nachrichten bzw. Anfragen pro Sekunde deutlich günstiger als alle anderen Hosting-Umgebungen sind (siehe 8.5.). Die Preise der Hosting-Umgebungen Azure App Service, Azure Container Instance und Azure VM bleiben konstant über die Zeitspanne von einem Monat und sind somit unabhängig von der Anzahl der Anfragen bzw. Nachrichten.

10.2 Interpretation der Ergebnisse

Die Hosting-Umgebung Azure Function verursacht in allen Auslastungsszenarien sowohl bei temporal gekoppelter als auch temporal entkoppelter Kommunikation mehr Scope 2- und 3-Emissionen als alle anderen Hosting-Umgebungen, ist aber dennoch bei niedriger Auslastung als auch bei mittlerer Auslastung verhältnismäßig deutlich günstiger. Das bedeutet, dass der Preis, welcher langfristig für die Verwendung der Azure Functions in Wirklichkeit bezahlt werden muss, die hohen

CO_2-äquivalenten Emissionen sind. Der Grund dafür ist jedoch nicht zwangsweise die Architektur der Azure Function. Es deutet viel mehr darauf hin, dass die Problemursache für die verhältnismäßig hohen CO_2-äquivalenten Emissionen der serverlosen Azure Function die automatische Skalierung bzw. deren Verhalten ist. Dies kann damit begründet werden, dass bereits bei dem niedrigen Auslastungsszenario von lediglich 2,5 Anfragen bzw. Nachrichten pro Sekunde deutlich mehr vCPU-Stunden anfallen, als es bei den anderen Hosting-Umgebungen der Fall ist (siehe 8.4.). Dies bedeutet, dass die automatische Skalierung der serverlosen Azure Function selbst bei einer niedrigen Auslastung mehr vCPUs reserviert als es vermutlich notwendig wäre, denn die prozentualen vCPU-Auslastungen dieser Hosting-Umgebungen sind mit unter 3 % sehr niedrig.

Eine weitere Eigenart der automatischen Skalierung ist, dass die vCPU-Stunden des Szenarios mit Azure Function und temporal entkoppelter Kommunikation bei der hohen Auslastung von 250 Nachrichten pro Sekunde deutlich höher sind, als es bei dem Szenario Azure Function mit temporal gekoppelter Kommunikation der Fall ist (siehe 8.4.). Dies wiederum hat zur Folge, dass sowohl Scope 2, als auch Scope 3-Emissionen verhältnismäßig deutlich ansteigen. Die Ursache des Problems liegt voraussichtlich in der hohen Latenz der Azure Cosmos DB (siehe 8.1.), welche sich wiederum in einer hohen Bearbeitungsdauer widerspiegelt (siehe 8.2.). Es ist möglich, dass die automatische Skalierung dies erkennt und deshalb die Azure Function bei temporal entkoppelter Kommunikation (Azure Service Bus) stärker parallelisiert, um den Gesamtdurchsatz aufrechtzuerhalten bzw. die Bearbeitungsdauer zu minimieren. Dies ist hochgradig kontraproduktiv, da davon auszugehen ist, dass die Ursache für die höhere Datenbanklatenz vermutlich die starke Parallelisierung und der damit parallele Zugriff auf die Datenbank ist. Dieses Problem lässt sich voraussichtlich durch die Provisionierung des Durchsatzes der Azure Cosmos DB lösen. Grundsätzlich ist das Verhalten der Azure Cosmos DB jedoch unerwartet, da die Dokumentation der Azure Cosmos DB explizit hervorhebt, dass die verwendete Serverlose Konfiguration der Datenbank zum einen resistent gegen spontane Lastspitzen und zum anderen besonders gut geeignet für Azure Functions ist [36]. Umgehen lässt sich die Problematik nur indem statt der Serverlosen Konfiguration eine Provisionierung des Durchsatzes veranlasst wird. Dies geht jedoch mit höheren Basiskosten einher und ist dementsprechend teurer, wenn die Auslastung über lange Zeiträume sehr gering ist. Nach einer Provisionierung des Durchsatzes ist davon auszugehen, dass sich das Skalierungsver-

halten der Azure Function mit temporal entkoppelter Kommunikation ähnlich dem Verhalten bei temporal gekoppelter Kommunikation verhält.

Microsoft hat sich bis 2030 zum Ziel gesetzt CO_2-negativ zu werden [18]. Ein Teil dieser Strategie beinhaltet den Kauf von CO_2-neutraler Energie [18]. Dementsprechend spielen in Zukunft die CO_2-äquivalenten Emissionen, welche durch den Energieverbrauch verursacht werden (Scope 2) keine Rolle mehr. Betrachtet man unter dieser Voraussetzung die gesamten CO_2-äquivalenten Emissionen der Szenarien, ist eine Minimierung der CO_2-äquivalenten Emissionen nur möglich durch die bestmögliche Nutzung der vorhandenen bzw. provisionierten Hardwareressourcen. Je höher die prozentuale vCPU-Auslastung einer provisionierten Ressource ist, desto effizienter ist demnach die Nutzung. Betrachtet man unter diesem Aspekt die prozentualen vCPU-Auslastungen der Szenarien bei niedriger und mittlerer Auslastung (2,5 und 25 Anfragen bzw. Nachrichten pro Sekunde) ist bei allen Hosting-Umgebungen eine massive Überprovisionierung zu beobachten (siehe 8.4.). Dies gilt entgegengesetzt der Erwartung auch für die serverlosen Azure Functions.

Unter dem Aspekt der Ressourceneinsparung sind die Hosting-Umgebungen Azure VM und Azure Container Instance problematisch, denn diese sind nicht automatisch Skalierbar. Dies hat zur Folge, dass im Falle von einem unterschiedlichen Lastverhalten über bspw. den Zeitraum von einem Tag, die Ressource so provisioniert werden muss, dass sie mit der maximalen Last umgehen kann. Dies wiederum hat zur Folge, dass zu der Zeit, in der die Ressource weniger stark belastet wird, viel inaktive vCPU-Stunden erzeugt werden, wodurch eine ineffiziente Nutzung der Cloud-Ressource entsteht.

Insgesamt zeichnet sich ab, dass die Wahl des Kommunikationsmusters bzw. der Kommunikationstechnologie keinen nennenswerten Einfluss auf CO_2-äquivalenten Emissionen hat. Allerdings vereinfacht die temporal entkoppelte Kommunikation über einen Message-Broker die horizontalen Skalierungsmöglichkeiten im Vergleich zur temporal gekoppelten Kommunikation. Auch in dem Punkt der Performance zeichnen sich keine wesentlichen Unterschiede ab. Technisch bedingt ist jedoch die temporal gekoppelte Kommunikation deutlich schneller in ihrem Antwortverhalten als die temporal entkoppelte Kommunikation. Für den Fall, dass der benutzenden Person einer Anwendung das Gefühl der direkten Manipulation suggeriert werden soll, sollte dementsprechend insbesondere bei hoher Last von der temporal entkoppelten Kommunikation abgesehen werden (siehe 8.3.).

10.3 Einschränkung bei der Interpretation der gebundenen Emissionen

Ein wesentliches Problem der Untersuchung ist fehlende Transparenz der Microsoft Azure Cloud. Im Falle des Cloud-Anbieters Microsoft Azure beginnt dies bereits damit, dass des sogenannte Emissions Impact Dashboard, ein alternatives Werkzeug zur Erfassung der CO_2-äquivalenten Emissionen, lediglich für Enterprise-Kunden verfügbar ist und dementsprechend im Rahmen dieser Untersuchung nicht verwendet werden konnte. [3].

Für die Hosting-Umgebungen Azure Function (Verbrauchsplan), Azure App Service und Azure Container Instance ist es nicht möglich die zugrundeliegende Hardware zu bestimmen. Dies hat zur Folge, dass der gesamte Durchschnitt des minimalen und maximalen Energieverbrauchs der möglichen Hardware, welche von Microsoft Azure verwendet wird, für die Energieberechnung verwendet wird. Dieser Durchschnitt wiederum ist aufgrund der Limitierung des Datensatzes sehr ungenau und die Methodik der Cloud Carbon Footprint-Software versucht bislang keine Abschätzung des Fehlers bei der Berechnung der Faktoren zu machen. Die Methodik der Cloud Carbon Footprint-Software geht außerdem von einer linearen Zunahme des Stromverbrauchs in Abhängigkeit der prozentualen vCPU-Auslastung aus. In der Realität ist dieses Verhalten jedoch nicht zwangsweise garantiert [47]. In Anbetracht der Ergebnisse relativiert sich jedoch diese Problematik dadurch, dass Microsoft in Zukunft auf CO_2-neutrale Stromquellen setzt. Wie bereits beobachtet (siehe 10.2.), führt dies wiederum dazu, dass der Stromverbrauch und die damit einhergehenden Scope 2-Emissionen keine signifikante Rolle mehr spielen.

Ein weitere sehr großes Problem der Szenarien, welche Azure Functions als Hosting-Umgebung verwenden, ist die Tatsache, dass es nicht eindeutig möglich ist die Anzahl der verwendeten vCPUs zu bestimmen. Die bisherigen Ergebnisse beruhen auf der Annahme, dass die Hosting-Instanzen auf einer isolierten VM ausgeführt werden und nach der Methodik der Cloud Carbon Footprint-Software deren vCPUs mit dem Verhältnis 2:1 auf reale Prozessorkerne abgebildet werden. Zusätzlich wird angenommen, dass darauf basierend die Anzahl der verfügbaren vCPUs über die von Application Insights bereitgestellten Daten berechnet werden können. Für den Fall, dass diese Annahmen falsch sind, sind die Ergebnisse der Szenarien mit der Hosting-Umgebung Azure Function als hinfällig zu betrachteten. Dies betrifft insbesondere die Scope 3-Emissionen.

10.4 Empfehlung für weiterführende Forschung

In Zukunft gilt es vorrangig zu untersuchen, wie die Ressourcen, die für die Ausführung von Azure Functions benötigt werden, auf die Hardware verteilt werden. Für den Fall, dass Azure Functions auf einer geteilten Ressource ausgeführt werden, besteht die Möglichkeit, dass insbesondere die Scope 3-Emissionen der Azure Functions deutlich geringer ausfallen könnten. Des Weiteren muss untersucht werden, ob die gewählte Ausführungsdauer von zwei Stunden für die serverlosen Azure Function zu kurz ist, denn es besteht die Möglichkeit, dass sich das Verhalten der automatische Skalierung nach längerer Zeit verändert.

11 Handlungsempfehlung

Basierend auf den Ergebnissen der Untersuchung ist nach dem aktuellen Stand von der Verwendung von serverlosen Azure Functions (Verbrauchsplan) abzuraten, da diese durch die automatische Skalierung zu viele Ressourcen reservieren.

Die Wahl des Kommunikationsmusters bzw. der Kommunikationstechnologie hat keine nennenswerten Auswirkungen auf die CO_2-äquivalenten Emissionen und kann dementsprechend frei gewählt werden.

Die Ergebnisse zeigen, dass die beste Möglichkeit zur Optimierung des CO_2-äquivalenten Fußabdrucks das Einsparen von provisionierten Cloud-Ressourcen ist. Dies bedeutet, dass inaktive Cloud-Ressourcen immer deprovisioniert werden sollten, um dem Cloud-Provider eine effiziente Zuteilung der Ressourcen zu ermöglichen, was letztendlich zu einer Optimierung der Aufteilung der Scope 3-Emissionen führt. Dementsprechend ist die horizontale Skalierung, insofern sie so konfiguriert werden kann, dass sie vorhandenen Ressourcen bestmöglich ausreizt, eine sehr gute Möglichkeit für die Optimierung der CO_2-äquivalenten Emissionen. Unter diesem Aspekt sollte von der Verwendung standardmäßiger Azure VMs und Azure Container Instances bei stark schwankender Auslastung abgesehen werden, da diese dann die provisionierte Ressource nicht effizient nutzen können. Microsoft Azure bietet jedoch Abhilfe für die Problematik. Statt standardmäßiger Azure VMs können Azure VM-Skalierungsgruppen und statt Azure Container Instances

kann der Azure Kubernetes Service (AKS) verwendet werden. Beide dieser Services unterstützen die horizontale automatische Skalierung.

Ein genereller Ansatz für die Einsparung von Cloud-Ressourcen ist die Bestimmung des maximalen Durchsatzes einer Anwendung in einem Stresstest. Anschließend kann der maximale reale Durchsatz der Anwendung mit den Ergebnissen des Stresstests verglichen werden. Falls dabei eine zu starke Diskrepanz erkennbar ist, sollte die vorhandene Ressource an die realen Begebenheiten angepasst werden.

Solange die Microsoft Azure-Cloud noch nicht vollständig mit erneuerbarer Energie betrieben wird, sollte außerdem ein Rechenzentrum in einem Land gewählt werden, welches eine niedrige CO_2-Intensität des lokalen Stromnetzes aufweist.

12 Fazit

Zusammengefasst untersucht die Bachelorarbeit die CO_2-äquivalenten Emissionen und setzt diese in Relation mit der Performance und Usability der einzelnen Bausteine einer verteilten Cloud-Anwendung an einer beispielhaften Web-Shop-Anwendung. Umgesetzt wird die Untersuchung durch die Simulation der Bausteine einer verteilten Cloud-Anwendung durch definierte Szenarien. Innerhalb dieser Szenarien werden Kommunikationsmuster, Kommunikationstechnologien und Hosting-Umgebungen permutiert. Dabei werden unterschiedliche Auslastungen der Anwendung durch definierte Auslastungsszenarien berücksichtigt. Durch Application Insights werden Messergebnisse erfasst und durch eine angepasste Methodik nach Cloud Carbon Footprint in CO_2-äquivalente Emissionen umgerechnet. Die berechneten CO_2-äquivalente Emissionen werden den Kosten, der Performance und Usability gegenübergestellt. Das Endergebnis der Gegenüberstellung ist die daraus resultierende Handlungsempfehlung an entwickelnde Personen.

Literatur

[1] Google. *CO2-Bilanz von Google Cloud reduzieren*. 2021. URL: https://cloud.google.com/architecture/reduce-carbon-footprint (besucht am 26. 07. 2022).

[2] Walshe N. *Microsoft Sustainability Calculator helps enterprises analyze the carbon emissions of their IT infrastructure*. 2020. URL: https://azure.microsoft.com/en-gb/blog/microsoft-sustainability-calculator-helps-enterprises-analyze-the-carbon-emissions-of-their-it-infrastructure/ (besucht am 26. 07. 2022).

[3] D. Mytton. „Assessing the suitability of the Greenhouse Gas Protocol for calculation of emissions from public cloud computing workloads“. In: *Journal of Cloud Computing* (2020). URL: https://journalofcloudcomputing.springeropen.com/articles/10.1186/s13677-020-00185-8 (besucht am 26. 07. 2022).

[4] Green Software Foundation. *Software Carbon Intensity Standard*. Version 1.0.0. Nov. 2021. URL: https://github.com/green-software-foundation/software_carbon_intensity (besucht am 26. 07. 2022).

[5] J. Ranganathan und P. Bhatia. *The Greenhouse Gas Protocol: a Corporate Accounting and Reporting Standard, Revised Edition*. März 2004, S. 116. ISBN: 1-56973-568-9.

[6] AWS. *Customer Carbon Footprint Tool*. o. J. URL: https://aws.amazon.com/de/aws-cost-management/aws-customer-carbon-footprint-tool/ (besucht am 26. 07. 2022).

[7] GCP. *CO_2-Bilanz*. o . J. URL: https://cloud.google.com/carbon-footprint (besucht am 26. 07. 2022).

[8] Microsoft. *Connect to the Emissions Impact Dashboard*. 2022. URL: https://docs.microsoft.com/en-us/power-bi/connect-data/service-connect-to-emissions-impact-dashboard (besucht am 08. 07. 2022).

[9] o. V. *Overview | Cloud Carbon Footprint*. o. J. URL: https://www.cloudcarbonfootprint.org/docs/overview (besucht am 08. 07. 2022).

[10] o. V. *Cloud Carbon Footprint*. o. J. URL: https://github.com/cloud-carbon-footprint/cloud-carbon-footprint (besucht am 11. 07. 2022).

[11] o. V. *Methodology | Cloud Carbon Footprint*. o. J. URL: https://www.cloudcarbonfootprint.org/docs/methodology (besucht am 09. 07. 2022).

[12] E. Sommer, M. Adler und J. et. al. Perkins. *Cloud Jewels: Estimating kWh in the Cloud*. 2020. URL: https://www.etsy.com/codeascraft/cloud-jewels-estimating-kwh-in-the-cloud/ (besucht am 11. 07. 2022).

[13] o. V. *ConsumptionTypes.ts*. o. J. URL: https://github.com/cloud-carbon-footprint/cloud-carbon-footprint/blob/trunk/packages/azure/src/lib/ConsumptionTypes.ts (besucht am 09. 07. 2022).

[14] Microsoft. *Azure-Computeeinheit (ACU)*. 2022. URL: https://docs.microsoft.com/de-de/azure/virtual-machines/acu (besucht am 31. 07. 2022).

[15] Microsoft. *App Service pricing*. o. J. URL: https://azure.microsoft.com/en-us/pricing/details/app-service/windows/ (besucht am 29. 07. 2022).

[16] Microsoft. *Dv5 and Dsv5-series*. 2022. URL: https://docs.microsoft.com/en-us/azure/virtual-machines/dv5-dsv5-series (besucht am 29. 07. 2022).

[17] Intel. *Intel® Xeon® Platinum Processor*. 2022. URL: https://www.intel.com/content/www/us/en/products/details/processors/xeon/scalable/platinum/products.html (besucht am 31. 07. 2022).

[18] Microsoft. *2021 Environmental Sustainability Report*. 2021.

[19] SPECpower Committee. *Power and Performance Benchmark Methodology V2.2*. 2014.

[20] Mytton D. *Cloud Carbon Coefficients*. 2021. URL: https://github.com/cloud-carbon-footprint/cloud-carbon-coefficients/blob/main/coefficients.ipynb (besucht am 01. 08. 2022).

[21] o. V. *Embodied Emissions*. o. J. URL: https://www.cloudcarbonfootprint.org/docs/embodied-emissions (besucht am 09. 07. 2022).

[22] M. Fowler. *CQRS*. 2011. URL: https://martinfowler.com/bliki/CQRS.html (besucht am 26. 07. 2022).

[23] Microsoft. *What is IaaS? Infrastructure as a service*. 2022. URL: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-iaas (besucht am 26. 07. 2022).

[24] Microsoft. *What is PaaS? Platform as a service*. 2022. URL: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-paas (besucht am 26. 07. 2022).

[25] Microsoft. *Serverless computing. An introduction to serverless technologies*. 2022. URL: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-serverless-computing (besucht am 26. 07. 2022).

[26] Microsoft. *Azure serverless. Go serverless—build apps faster without managing infrastructure*. 2022. URL: https://azure.microsoft.com/en-us/solutions/serverless (besucht am 26. 07. 2022).

[27] R. T. Fielding. „Architectural Styles and the Design of Network-based Software Architectures“. Diss. University of California, Irvine, 2000.

[28] gRPC Autoren. *gRPC over HTTP2*. 2022. URL: https://github.com/grpc/grpc/blob/master/doc/PROTOCOL-HTTP2.md (besucht am 26. 07. 2022).

[29] The GraphQL Foundation. *Serving over HTTP*. 2022. URL: https://graphql.org/learn/serving-over-http/#gatsby-focus-wrapper (besucht am 26. 07. 2022).

[30] Microsoft. *Competing Consumers pattern*. o. J. URL: https://docs.microsoft.com/en-us/azure/architecture/patterns/competing-consumers (besucht am 26. 07. 2022).

[31] Microsoft. *Queue-Based Load Leveling pattern*. o. J. URL: https://docs.microsoft.com/en-us/azure/architecture/patterns/queue-based-load-leveling (besucht am 26. 07. 2022).

[32] Microsoft. *What is Azure Service Bus?* 2022. URL: https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-messaging-overview (besucht am 26. 07. 2022).

[33] Microsoft. *Ressourcenverfügbarkeit für Azure Container Instances in Azure-Regionen*. 2022. URL: https://docs.microsoft.com/de-de/azure/container-instances/container-instances-region-availability (besucht am 29. 07. 2022).

[34] Microsoft. *Azure Functions-Hostingoptionen*. 2022. URL: https://docs.microsoft.com/de-de/azure/azure-functions/functions-scale (besucht am 30. 07. 2022).

[35] Microsoft. *Konzepte für Azure Functions-Trigger und -Bindungen*. 2022. URL: https://docs.microsoft.com/de-de/azure/azure-functions/functions-triggers-bindings?tabs=csharp (besucht am 30. 07. 2022).

[36] Microsoft. *Azure Cosmos DB serverlos*. 2022. URL: https://docs.microsoft.com/de-de/azure/cosmos-db/serverless (besucht am 10. 08. 2022).

[37] Microsoft. *Application Insights-Übersicht*. 2022. URL: https://docs.microsoft.com/de-de/azure/azure-monitor/app/app-insights-overview (besucht am 02. 08. 2022).

[38] Microsoft. *Unterstützte Sprachen*. 2022. URL: https://docs.microsoft.com/de-de/azure/azure-monitor/app/platforms (besucht am 02. 08. 2022).

[39] Microsoft. *Systemleistungsindikatoren in Application Insights*. 2022. URL: https://docs.microsoft.com/de-de/azure/azure-monitor/app/performance-counters (besucht am 05. 08. 2022).

[40] Microsoft. *Protokollbasierte Metriken von Application Insights*. 2022. URL: https://docs.microsoft.com/de-de/azure/azure-monitor/essentials/app-insights-metricss (besucht am 05. 08. 2022).

[41] Microsoft. *Abhängigkeitsnachverfolgung in Azure Application Insights*. 2022. URL: https://docs.microsoft.com/de-de/azure/azure-monitor/app/asp-net-dependencies (besucht am 06. 08. 2022).

[42] Microsoft. *Übersicht über die Kusto-Abfragesprache (KQL)*. 2022. URL: https://docs.microsoft.com/de-de/azure/data-explorer/kusto/query/ (besucht am 06. 08. 2022).

[43] Microsoft. *General purpose virtual machine sizes*. 2022. URL: https://docs.microsoft.com/en-us/azure/virtual-machines/sizes-general (besucht am 27. 08. 2022).

[44] European Environment Agency. *Greenhouse gas emission intensity of electricity generation in Europe*. o. J. URL: https://www.eea.europa.eu/ims/greenhouse-gas-emission-intensity-of-1 (besucht am 26. 07. 2022).

[45] Microsoft. *Preisrechner*. o. J. URL: https://azure.microsoft.com/de-de/pricing/calculator/ (besucht am 27. 08. 2022).

[46] Microsoft. *Azure Functions pricing*. o. J. URL: https://azure.microsoft.com/en-us/pricing/details/functions/ (besucht am 27. 08. 2022).

[47] B. Davy. *Estimating AWS EC2 Instances Power Consumption*. 2021. URL: https://medium.com/teads-engineering/estimating-aws-ec2-instances-power-consumption-c9745e347959 (besucht am 26. 07. 2022).