Evaluating the Energy Consumption Impact of a Carbon Aware Autoscaling in Microservice-Based Applications on the Public Cloud

A Sustainability Perspective

Lappeenranta-Lahti University of Technology LUT

LUT School of Engineering Sciences

Master of Science (Tech.) in Software Engineering, Master's thesis

2023

Haben Birhane Gebreweld

Examiners: Professor Jari Porras (LUT University)

Professor Henry Muccini (University of L'Aquila)

Professor Patricia Lago (Vrije Universiteit Amsterdam)

SE4GD
Software Engineers for Green Deal

With the support of the
Erasmus+ Programme
of the European Union

This thesis has been accepted by partner institutions of the consortium (619839-EPP-1-2020-1-FI-EPPKA1-JMD-MOB).

Successful defence of this thesis is obligatory for graduation with the following national diplomas:

- Master of Computer Science (University of L'Aquila)
- Master of Science in Technology (LUT University)
- Master of Computer Science (Vrije Universiteit Amsterdam)

ABSTRACT

Lappeenranta–Lahti University of Technology LUT

LUT School of Engineering Science

Software Engineering

Haben Birhane Gebreweld

Evaluating the Energy Consumption Impact of a Carbon Aware Autoscaling in Microservice-Based Applications on the Public Cloud: A Sustainability Perspective

Master's thesis

2023

76 pages, 26 figures, 8 tables and 0 appendices

Advisor & Co-Supervisors: Professor Henry Muccini (University of L'Aquila)
Dr. Karthik Vaidhyanathan (IIT Hyderabad)
Roberta Capuano (University of L'Aquila)

Examiners: Professor Jari Porras (LUT University)
Professor Henry Muccini (University of L'Aquila)
Professor Patricia Lago (Vrije Universiteit Amsterdam)

Keywords: Carbon Aware Autoscaling, Green Autoscaling, Carbon Efficiency, Kubernetes Event Driven Autoscaling(KEDA), Carbon Intensity, Microservices, Green Cloud

Background: The growing awareness of the environmental impact of the ICT industry, including the software sector, and the emergence of sustainability policies and compliance requirements have fueled an increasing interest in sustainable application development and operation. Cloud computing has been widely acclaimed for its scalability, presenting prospects for sustainable software operation. Nonetheless, existing research is deficient in

carbon-aware scaling strategies, especially in the context of microservice-based applications. Bridging this gap can lead to the establishment of environmentally conscious and efficient application scaling methods. This endeavor holds promise for advancing sustainability in the field of cloud computing.

Aim: The aim of this research is to empirically evaluate the energy consumption impact of Carbon-Aware Autoscaling on microservice-based applications in the Public Cloud.

Method: In this study, we conducted an empirical experiment to evaluate the impact of the autoscaling strategy on a microservice-based application. The experiment was designed to collect data on two dependent variables: energy consumption and response time. Through this approach, we aimed to gain insights into how the chosen autoscaling strategy affects these metrics.

Result: Our analysis resulted in three key discoveries regarding the impact of Carbon Aware Autoscaling on microservice-based applications. Primarily, it presents substantial energy cost efficiency, making it particularly advantageous in handling larger volumes of microservices. Secondly, Carbon Aware Autoscaling demonstrates notably lower energy consumption compared to HPA-based autoscaling, reducing energy usage by 42.9% on average on application level and 38.24% on average on microservice level, especially advantageous for microservices with heavy workloads. Thirdly, the autoscaler significantly impacts response time, showing an average increment of 30.68%, making it well-suited for low-priority workloads that do not require real-time processing. These findings provide essential insights for future research in the field of autoscaling strategies for microservice-based applications.

Conclusion: our analysis suggests the following insights: **a)** Extending the experiment to workload scheduling based on spatial and temporal carbon intensity values could yield valuable findings. **b)** Validating the effectiveness of the autoscaling approach with real-world microservices-based applications that do not require real-time processing is recommended. **c)** Mixing the two autoscaling strategies based on microservice workload may further improve energy consumption reduction and mitigate the impact on application response time. These recommendations offer potential avenues for future research and optimization in the field of autoscaling strategies for microservices-based applications.

ACKNOWLEDGMENTS

The past two years have presented immense challenges in my life, as my home country has been devastated by a genocidal war, and I faced the distressing uncertainty of not knowing the whereabouts of my family for approximately 8 months. Despite these trying circumstances, I persevered and remained focused on my studies, pushing myself to the limits to complete my academic journey. Throughout this difficult period, I am deeply grateful for the unwavering support and care provided by my friends and colleagues, who stood by me and helped me navigate through the setbacks. Their presence and encouragement have been invaluable, and I extend my heartfelt thanks to each one of them.

I extend my heartfelt gratitude to my thesis advisor, Professor Henry Muccini, and my co-supervisors Dr. Karthik Vaidhyanathan and Roberta Capuano for their invaluable guidance and profound contributions throughout the thesis process. I am also deeply thankful to the SE4GD program coordinators, Professor Jari Poras, Professor Henry Muccini, and Professor Patricia Lago, for providing me with this incredible opportunity to be part of this program. Their unwavering support and encouragement have been pivotal in shaping my academic journey and research endeavors. Special appreciation goes to Susanna for her exceptional coordination and assistance.

Lastly, I want to give special recognition to my family. Despite the challenges and uncertainties in our home country over the past two years, your unwavering love and support have kept me going. Your belief in me, words of encouragement, and sacrifices have been my driving force. This achievement wouldn't have been possible without you. A special mention goes to my Mom, who has worked tirelessly to ensure my success, without expecting anything in return. Even though you couldn't be here to celebrate with me, I want you to know how profoundly grateful I am.

SYMBOLS AND ABBREVIATIONS

Roman / Greek characters

$\forall$	for all
$\exists$	there exists
μ	mean
ρ	population correlation coefficient
e	energy
r	response time
m	microservices
c	subsets of carbon aware components

Abbreviations

AKS Azure Kubernetes Services

ANOVA Analysis of Variance

API Application Program Interface

CLI Command Line Interface

CPU Central Processing Unit

DNS Domain Name Service

ePBF extended Berkeley Packet Filter

GHG Greenhouse Gases

GQM Goal-Question-Model

HPA Horizontal Pod Autoscaler

IaaS Infrastructure as a Service

ICT Information and Communication Technologies

KEDA Kubernetes Event-Driven Autoscaling

KEPLER Kubernetes Efficient Power Level Exporter

PaaS Platform as a Service

QOS Quality Of Service

SaaS Service as a Service

SDK Software Development Kit

SLA Service Level Agreement

SOA Service-Oriented Architecture

SSE Sustainable Software Engineering

SusAF Sustainability Awareness Framework

VM Virtual Machine

YAML Yet Another Mark-up Language

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 Introduction

In today's world, sustainability is not merely a choice but a vital necessity. The growing concerns about climate change and its environmental effects have made it critical to take action to reduce our impact on the planet. Additionally, new policies and compliance regulations[1] in various regions have sparked an increasing interest in sustainability among industries and organizations. In the software industry, there is also a rising momentum towards sustainable application development. As the number of applications grows, so does the awareness of their environmental impact on the underlying infrastructure [2]. The rise of Cloud-native development, with its dynamic scaling and Kubernetes-based technologies, is enabling efficient workload and resource management. Beyond their historical use for cost management and accelerated development[3], these cloud-native approaches now present avenues for optimizing application development to enhance sustainability.

Cloud autoscaling, despite being in its early stages, has seen the development and adoption of various widely utilized approaches in the industry. These approaches include cost-aware, load-aware, resource-aware, data-aware, Quality Of Service (QOS)-aware, Service Level Agreement (SLA)-aware, and budget-aware autoscaling approaches[4]. In recent times, due to increased awareness of the environmental impact of the Information and Communication Technologies (ICT) industry[5], Carbon Aware approaches are becoming more prominent and gaining momentum in autoscaling and scheduling of workloads.[6][7]

The term "carbon aware software" refers to a software that adapts its behavior based on the carbon intensity of the energy it consumes[8] according to the definition given by Green Software Foundation[1] for Software Carbon Intensity technical specification. In cloud environments, horizontal scaling[2] is a commonly defined practice, but it not carbon aware. Horizontal scaling typically has a maximum scaling capacity specified in the Scaler definition, which is often constrained by resource availability and budgetary limit of the software

[1] `https://greensoftware.foundation/`

[2] `https://www.techopedia.com/definition/7594/horizontal-scaling`

owners, and does not adjust it behaviour in response to changes in the carbon intensity of the energy it consumes.

This thesis presents the implementation and empirical experimentation of both Horizontal Pod Autoscaler (HPA) and Carbon Aware Autoscaler on a microservice-based web application deployed in the public cloud. The objective is to investigate the influence of carbon aware autoscaling on the energy consumption and response time of the microservice-based application, as compared to the HPA-based autoscaling.

The study provides significant contributions, including:

- ✓ Practical implementation of Carbon Aware Autoscaling for microservice-based applications on the public cloud.
- ✓ Empirical experimentation of Horizontal Pod Autoscaler (HPA) and Carbon Aware Scaler, analyzing energy consumption and response time.
- ✓ Development of reusable Read Intensive and Write Intensive workload scenarios for the Sock Shop microservice-based web application.
- ✓ Detailed energy consumption measurement at a granular level (energy consumption per pod) using Kepler to assess the autoscaler's impact on individual pods.

The thesis is organized as follows: Chapter 2 provides the research context with background information. Chapter 3 discuses the problem with the current autoscalers in detail and proposes a solution. Chapter 4 outlines the experiment definition and planning. In Chapter 5, we discuss the execution of the experiment. We present a summary of experiment results, addressing each research question and associated hypotheses using statistical analysis in Chapter 6, followed by a detailed analysis of the findings in Chapter 7. The analysis aims to make sense of the results and convey the main insights gained from the study. Chapter 8 thoroughly discusses the threats to the validity of our study. Finally, Chapter 2.4 reviews previous studies that have explored Horizontal Pod Autoscaling and Carbon awareness in the cloud, and Chapter 9 provides our conclusion.

2 Background and Related Works

This research focuses on evaluating the impact of Carbon-Aware Autoscaling on reducing the energy consumption of microservice-based applications deployed on the public cloud. This topic holds enormous significance in the context of sustainable software engineering and cloud computing.

2.1 Sustainable Software Engineering

Sustainable Software Engineering (SSE) is an emerging discipline at the intersection of climate science, software, hardware, electricity markets, and data-center design. It encompasses core philosophies and principles necessary to define, build, and run sustainable software applications [9]. Sustainable Software Engineering combines two core philosophies and six principles in tandem[10][11].These six principles are independent of application domain, organization size or type, cloud vendor or self-hosted, and programming language or framework.[12]

These philosophies are:

- **Collective responsibility for climate action:** Sustainable Software Engineering places importance on inclusivity by acknowledging that everyone, irrespective of their sector, industry, role, or technology, can play a role in tackling climate challenges. It emphasizes that each individual can make a significant impact.[10]
- **Sustainability as the intrinsic motivation:** Developing sustainable applications offers multiple benefits such as cost-effectiveness, enhanced performance, and increased resilience. However, the primary motivation behind Sustainable Software Engineering is the pursuit of sustainability itself, with other advantages being secondary considerations. [10]

The Six principles of Sustainable Software Engineering are essential across all domains[13] and discussed thoroughly below:

1. **Carbon Efficiency:** The first principle of Sustainable Software Engineering is to develop applications that minimize carbon emissions, known as carbon efficiency. Greenhouse Gases (GHG) contribute to an increase in Earth's temperature, mainly due to human activities. GHG quantities are converted to carbon dioxide equivalent (CO2eq) to simplify measurements. For instance, one ton of methane is equivalent to approximately 25 tons of CO2, so it is represented as 25 tons CO2eq.[14] In some cases, the term "carbon" is used to encompass all GHGs, streamlining the discussion. The objective is to optimize the value derived from each gram of carbon emitted into the atmosphere, acknowledging that carbon emissions are inevitable in our activities. Being carbon-efficient involves reducing the amount of carbon released per unit of work.[15] In other words, it means *creating applications* that provide ***equivalent value*** to you or your users while *emitting* a ***lesser amount of carbon.***

2. **Energy Efficiency:** Electricity is convenient, but it is mostly generated by burning fossil fuels, which is the biggest contributor to carbon emissions.[1] Electricity can be considered one of the proxies for carbon. Software, whether it runs on smartphones or in data centers for training machine learning models, requires electricity for its operation. A highly effective approach to minimizing electricity consumption and the resulting carbon emissions from our software is to enhance the energy efficiency of our applications. This is why building energy-efficient applications is considered the second principle of sustainable software engineering.[16][17]

3. **Carbon Awareness:** Electricity production varies based on different locations and times, utilizing diverse sources with varying carbon emissions. Certain sources, like wind, solar, or hydroelectric, are considered clean and renewable, emitting no carbon during generation. However, other sources may have different environmental impacts, resulting in varying carbon emissions associated with electricity production.[18]

   Carbon Intensity is a measure of how many carbon (CO2eq) emissions are produced per kilowatt-hour of electricity consumed. Electricity generation methods

[1] `https://www.eea.europa.eu/data-and-maps/daviz/change-of-co2-eq-emissions-2\#tab-chart_4`

vary across different locations and time periods, resulting in varying carbon emissions. Certain sources, such as wind, solar, and hydroelectric power, are considered clean and renewable, emitting no carbon during electricity production. Conversely, fossil fuel sources exhibit different levels of carbon emissions. For instance, gas-burning power plants generally release less carbon compared to coal-burning power plants.[18][19]

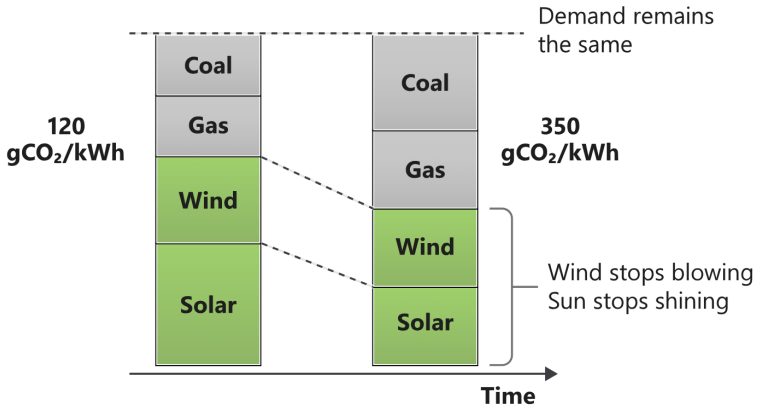


Figure 1: Carbon Intensity Visualization: Image used with permission from Microsoft Learn[2], sourced from the module on "Principle: Carbon Awareness"[18]

Carbon intensity varies across different locations as the energy mix differs, with some regions incorporating a higher proportion of clean energy sources compared to others. Furthermore, carbon intensity fluctuates over time due to the inherent variability of renewable energy. For instance, when there are cloudy conditions or minimal wind, carbon intensity tends to rise as a larger portion of the electricity mix originates from carbon-emitting sources.[19]

4. **Hardware Efficiency:** This term "Embodied Carbon" refers to the carbon emissions that are produced during the entire lifecycle of hardware components, including computer servers, networking equipment, and devices. This includes the energy and resources used during the extraction of raw materials, manufacturing, transportation, use, and disposal or recycling processes.[20] Minimizing the environmental impact of the technology ecosystem is critical for sustainable software engineering. One

way to be more hardware efficient is to extend the lifespan of hardware, as computers do not wear out; they only become obsolete due to constantly updated software that pushes their limits.[21]

5. **Measurement:** The greenhouse gas accounting standard classifies emissions into three distinct scopes[22]:

    - Scope 1 encompasses direct emissions arising from an organization's owned or controlled operations, such as on-site fuel combustion or fleet vehicles.
    - Scope 2 covers indirect emissions related to purchased energy, like heat and electricity generation.
    - Scope 3 represents other indirect emissions originating from a wide array of activities, including an organization's supply chain, business travel, and electricity consumption by customers.

    Scope 3, or value chain emissions, are significant and complex for organizations to calculate. They cover all activities from product conception to distribution.[23]

6. **Climate Commitments:** Numerous methods exist to lower emissions. It is crucial to comprehend the specific reduction approach when aiming to achieve emission reduction targets. At highest level, it can be achieved through offsets or abatement mechanisms. Offsetting involves reducing emissions elsewhere, while reduction or elimination focuses on avoiding carbon emissions. Since complete elimination of carbon emissions is challenging in modern society, offsets offer a way to complement reduction efforts.[24] Offsets can be achieved through compensation or neutralization. Compensation, also known as avoidance, entails paying others not to emit carbon. On the other hand, neutralization involves removing carbon from the atmosphere, such as through enhancing natural carbon sinks or employing direct air capture techniques.[25]

2.2 Cloud Autoscaling

Cloud computing has become a widely-used and efficient technology. Scalability is key to success for cloud-based businesses. Auto scaling enables the adjustment of cloud resources in response to changes in demand. By improving fault tolerance, availability, and cost management, auto scaling can significantly improve an organization's service level agreements, resulting in better performance and reliability.[26]

The autoscaling feature in cloud environments enables automatic adjustment of resource capacity for applications. Auto-scaling is widely available as a desirable feature in cloud Infrastructure as a Service (IaaS) and Platform as a Service (PaaS) offerings provided by various cloud vendors. It allows for seamless and dynamic scaling of resources based on application demands, ensuring optimal performance and resource utilization.[27]

2.2.1 Autoscaling Policies

Scaling an application can be achieved through two primary policies:[28]

- **Vertical Autoscaling**, also known as scaling up and down, involves adjusting the capacity of a resource. [29] For instance, migrating an application to a larger Virtual Machine (VM) size. Vertical scaling often requires temporarily making the system unavailable during redeployment, making it less common to automate this type of scaling.
- **Horizontal Autoscaling**, also referred to as scaling out and in, focuses on adding or removing instances of a resource. The application continues running seamlessly as new resources are provisioned. Once the provisioning process is completed, the solution is deployed across these additional resources.[30] In case of reduced demand, the additional resources can be efficiently shut down and deallocated without interrupting the application's operation.

2.2.2 Autoscaling Components

Typically, a cloud autoscaling strategy includes the following components[31]:

- Instrumentation and monitoring systems at the application, service, and infrastructure levels. These systems capture important metrics such as response times, queue lengths, Central Processing Unit (CPU) utilization, and memory usage.
- Decision-making logic that evaluates these metrics against predefined thresholds or schedules and decides whether to scale.
- Components that enable the system to scale.
- Regular testing, monitoring, and tuning of the autoscaling strategy to ensure that it works as intended.

2.2.3 Types of Autoscaling

In a nutshell, autoscaling can be of the following types[32]:

1. **Reactive Autoscaling**: In a reactive autoscaling approach, resources are adjusted dynamically in response to sudden increases in traffic. This method relies on continuous real-time monitoring of available resources.[33] A cooldown period is often implemented to handle potential future traffic spikes. During this period, resources are maintained at maximum capacity, even if the traffic temporarily decreases. The purpose is to handle any further unexpected surges in traffic effectively.
2. **Predictive autoscaling**: This approach leverages machine learning and artificial intelligence techniques to assess traffic patterns and forecast resource requirements. Predictive models are employed to analyze historical workload data and anticipate future resource needs for the upcoming days.[34][35] Scheduled scaling actions are then performed based on these predictions, ensuring that sufficient resource capacity is available before it is required by the application.[34] In essence, predictive autoscaling utilizes predictive analytics, past usage data, and current usage patterns to automatically scale resources in alignment with projected future demands.[36][37]
3. **Scheduled autoscaling**: This approach allows users to define specific time ranges for adding additional resources. Scheduled scaling works best when there are predictable fluctuations in traffic at specific times of the day, although these changes

are typically sudden.[38] For instance, resources can be provisioned in advance for a significant event or a peak period throughout the day, eliminating the need to scale up resources as demand increases. This proactive approach ensures readiness and avoids any potential delays in resource scaling. The approach combines real-time monitoring, prediction of anticipated traffic changes, and pre-defined intervals for resource adjustments.

4. **Manual Scaling**: With manual scaling, you can manually adjust the number of instances either through a CLI or console. This is a great option if automatic scaling isn't necessary for your users.[32]

5. **Dynamic Scaling**: Dynamic scaling is another type of auto scaling where the number of instances changes automatically based on received signals. This is a great option when dealing with a high volume of unpredictable traffic.[39]

2.3 Microservice Architectural Style

The profusion of impactful publications and case studies centered around microservice-based architectures, exemplified by Lewis and Fowler's seminal work in 2014 Lewis and Fowler[40], and the valuable insights shared by Cockroft[41] during a Silicon Valley Microservices Meetup, likely played a significant role in driving the broad acceptance and adoption of microservices within the industry.

At the heart of microservice-based architecture lies the concept of building complex applications by composing small, loosely coupled components that encapsulate specific business capabilities and communicate through language-agnostic Application Program Interface (API), inspired by Service-Oriented Architecture (SOA) principles[42]. This architectural style has gained significant popularity in cloud-based Service as a Service (SaaS) offerings, offering advantages such as enhanced software development agility and improved scalability of deployed applications [40][43]. According to Beck et al.[44], breaking down large applications into individual microservices fosters greater independence for agile teams , reducing synchronization efforts and enabling each team to autonomously deploy changes to production as per their schedule.

Microservice-based architectures offer advantages like independent deployment and individual scaling based on increasing traffic[45][46] [47] [48]. This approach enhances fault-tolerance, preventing defective microservices from disrupting the entire application [49] [50][46] [51]. These characteristics are crucial for achieving carbon-aware scaling, ensuring resource efficiency and sustainability in cloud-based systems.

2.4 Related Works: Carbon Awareness, Microservices & Cloud Autoscaling

The related work section explores various initiatives aimed at achieving sustainability in cloud computing, with a specific focus on Carbon Aware autoscaling. Although this area is still in its early stages, we have identified a limited number of published papers in the last year and a half. In the following review, we examine pioneering works that have addressed challenges in this domain, analyze their contributions, establish connections to our current study, and highlight any points of differentiation. By delving into these existing efforts, we aim to contextualize our research within the broader landscape of sustainable cloud computing practices.

Observing, the existence of a noticeable gap in addressing both performance and energy aspects to support the execution of microservices applications in the cloud, Nardin et al.[52] proposed ”Elergy,” a lightweight proactive elasticity model designed for cloud-based microservices applications. Elergy adopts a differential approach by periodically reorganizing resources to execute the application with an optimal amount of resources while maintaining or improving the performance of CPU-bound applications. Their research demonstrated a reduction in energy consumption and competitive cost when compared to a non-elastic scenario. Our study shares a common goal with the aforementioned study as we also aim to enhance the energy consumption of microservices-based applications in the public cloud. However, our approach differs in that we explore the use of carbon aware autoscaling to achieve this objective, instead of periodically adjusting resource allocation. Moreover, in addition to energy consumption, our empirical assessment also focuses on analyzing the response time of the microservices-based application, providing a comprehensive evaluation of its performance.

The challenge of optimizing energy consumption and reducing carbon emissions in cloud computing is a hot topic for researchers nowadays. Recently, James et al.[7] focused on demand-side management by implementing a low carbon scheduling policy in Kubernetes. They migrated energy consumption to regions with the lowest carbon intensity of electricity, aiming to minimize carbon emissions. Their study provided a generic model for workload placement optimization based on carbon intensity and evaluated its performance in a case study with a major public cloud provider. Similarly, our study also leverages carbon intensity data to optimize energy consumption through autoscaling. We assess the performance of our approach in a case study on a public cloud. However, a key difference lies in the objective: we determine how far to scale based on carbon intensity, whereas their approach involves spatially moving the workload to regions with low carbon intensity. Both studies contribute to the broader goal of sustainability in cloud computing, addressing the challenge of reducing environmental impact.

With the growing popularity of temporal shifting of workloads in cloud platforms to utilize greener electricity supply, a new challenge has emerged, leading to extended job completion times due to the suspend-resume approach. In response, Hanafy et al.[6] proposed the concept of carbon scaling, dynamically adjusting job server allocations based on fluctuations in the carbon cost of electricity supply. They developed CarbonScaler, an optimal greedy algorithm to minimize job emissions, and successfully implemented it in Kubernetes using autoscaling capabilities. Their evaluation demonstrated significant carbon savings, achieving up to 50% compared to carbon agnostic execution and up to 35% improvement over suspend-resume policies. Our study aligns with Hanafy et al. in leveraging Kubernetes autoscaling based on carbon intensity data. However, we differentiate ourselves by focusing on microservice-based web application workloads, with a significant emphasis on analyzing the impact on response time, unlike their focus on non-real-time workloads. Additionally, our main focus is on energy efficiency, while their emphasis lies primarily on carbon efficiency.

3 Study Design

This chapter presents the study design, which includes the statement of the problem, formal definition of the study, and the methodology employed to address the research questions.

3.1 Statement of Problem

Autoscaling offers users an automated method for dynamically adjusting the allocation of compute, memory, or networking resources in response to fluctuations in traffic and usage patterns[53]. As previously discussed (see Chapter 2), autoscaling encompasses two policies: Horizontal and Vertical Autoscaling. Historically, autoscaling primarily aimed at cost management and performance optimization. By operating in a dynamic environment, autoscaling effectively avoids resource underutilization or overprovisioning. As depicted in Figure 2, the horizontal autoscaler functions by creating additional software instances in response to increased workload, and scales down when workload decreases.

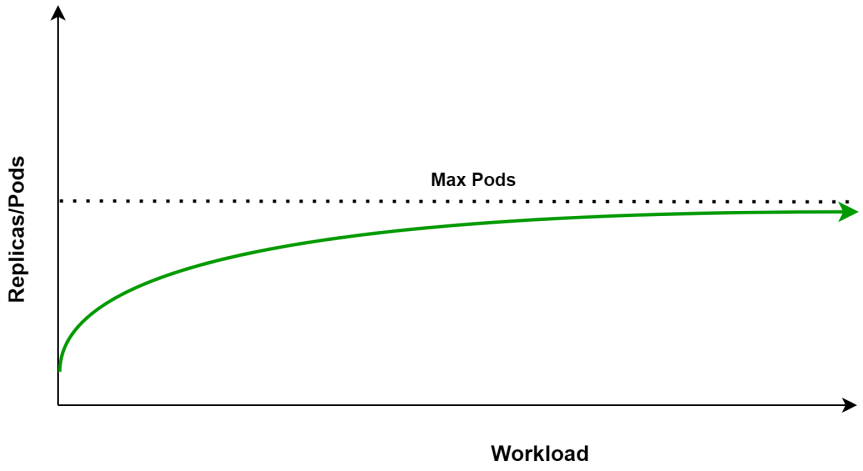
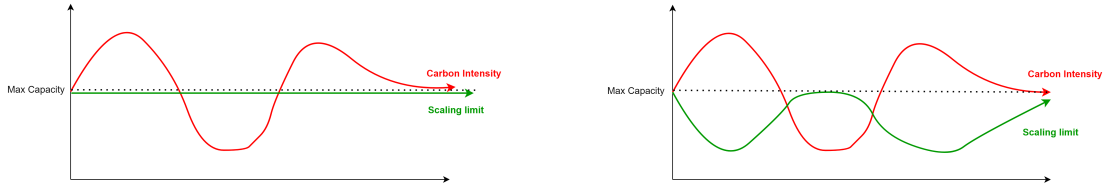


Figure 2: Visual Representation of how horizontal scaling works with workload

The horizontal autoscaling approach efficiently manages workloads and responds to demands on the software system. However when we evaluate it in terms of sustainability, it assumes a constant carbon intensity of the energy used, leading to a fixed level of carbon emissions. In reality, the carbon intensity can vary spatially (based on deployment location) and temporally (time of the day)[54]. In traditional scalers with horizontal autoscaling policy (Figure 3(a)), the limit to scaling capacity remains constant and independent of carbon intensity. On the other hand, with carbon aware scaling (Figure 3(b)), the scaling capacity changes in response to the carbon intensity value, allowing the software to scale more during low carbon intensity periods and scale less during high carbon intensity periods, making scaling decisions based on both resource demand and carbon intensity.



(a) Horizontal Pod Scaling (b) Carbon Aware Scaling

Figure 3: visualization of how the scaling capacity of software changes for both Horizontal Pod scaling and Carbon Aware Scaling

3.2 Study Definition

We used the **Goal-Question-Model (GQM)** approach to clearly define the thesis as proposed by Basili et al.[55] as follows:

Investigate	Carbon-Aware KEDA Autoscaling
For the purpose of	Energy Consumption Evaluation
With respect to	Microservice based applications
From point of view of	Sustainable Software Engineer
In the context of	Public Cloud

The **Goal** of the thesis is:

To *Investigate* ***Carbon-Aware KEDA Autoscaling*** *for the purpose of* ***Energy Consumption Evaluation*** *with respect to* ***Microservice-Based Applications*** *from point of view of* ***sustainability*** *in the context of* ***Public Cloud***.

In order to accomplish the aforementioned Goal, we identified two primary domains that could potentially be influenced, namely energy consumption and performance. As a result, we formulated three main research questions to further refine our goal:

1. What is the relationship between the number of microservices and the energy consumption overhead imposed by Carbon Aware KEDA Autoscaling components in the public cloud?

   In comparison to the HPA-based autoscaling, the Carbon Aware KEDA autoscaling incorporates additional components to facilitate its intended functionality. This research question aims to explore the energy consumption overhead caused by these components and how this impact varies with changes in the number of microservices and workloads.

2. How does the Carbon-Aware KEDA Autoscaling compare to HPA-Based Autoscaling strategies in terms of energy consumption?

   In order to evaluate the influence of carbon-aware autoscaling on energy consumption of the microservice based application, our approach involves deploying two distinct versions of a microservice-based application, each utilizing a different Autoscaling strategy. Through execution on identical workload scenarios, we will collect energy consumption data for both instances. Subsequently, a comparative analysis will be conducted to identify the specific impact of the carbon-aware Autoscaling strategy on energy consumption.

3. How does the carbon-aware KEDA Autoscaling impact the response time of the microservice based application in the public cloud?

   (a) Is there a statistically significant correlation between the energy consumption and response time of the microservice-based application when using eirher of the autoscalers?

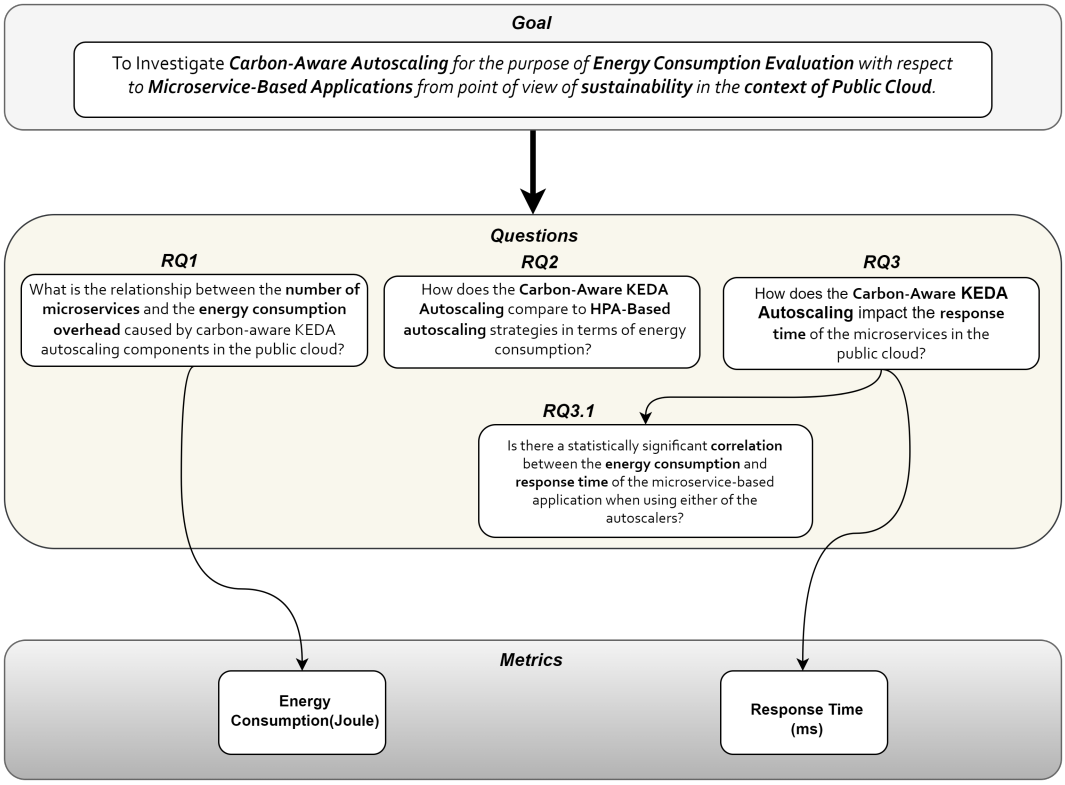


Figure 4: Visual Representation of the GQM

Performance is a key aspect to examine, and to analyze the performance impact of the scaling strategy, we will primarily focus on gathering performance test data, specifically response time, for each strategy. Subsequently, we will conduct comparative tests to assess the performance costs associated with the scaling strategies. And finally we will investigate if there is any correlation between the energy consumption and response time of the microservice based application.

For this purpose we have identified the following metrics:

- **Energy Consumption**: To assess the energy consumption of pods[1] within a specific namespace of an application deployed on a public cloud, we will employ **Ke-**

[1] `https://kubernetes.io/docs/concepts/workloads/pods/`

pler[2] Kubernetes Efficient Power Level Exporter (KEPLER) as our measurement tool. This approach will enable us to gather energy consumption data on granular level(pod level) as it runs in Kubernetes cluster and capable of collecting energy consumption of Kubernetes components(Pods, Nodes).

- **Performance Metrics**: We will mainly collect the response time of the application from **Locust**[3] performance testing tools.

3.3 Methodology

We provide a five-phase research methodology in order to address the RQs. Figure 5 shows an overview, which is expanded upon below.

In **Step (1)** of the experiment, a rigorous selection process was employed to identify a suitable subject for deployment on a Kubernetes cluster. Ultimately, the chosen subject was the Sock Shop: A Microservice Demo Application(Table 1), a renowned open-source application licensed under Apache License, Version 2.0, and freely available for testing and demonstration purposes.

Public cloud providers exhibit variances across a range of factors such as computing and storage capabilities, availability, security, accessibility, adaptability, guaranteed scalability, interoperability, and cost. These distinctions contribute to their comparative positioning and necessitate careful evaluation when selecting a provider[56]. Hence, we have opted to restrict our experiment to a single public cloud service provider, specifically Azure. Azure currently possesses the second largest[4] share in the global cloud service market, offering an extensive array of regions for provisioning, alongside a multitude of cloud-native services. Among the various container orchestration tools available, our choice has landed on Kubernetes[5] due to its exceptional popularity[6], open-source nature, and substantial de-

[2] `https://sustainable-computing.io/`

[3] `https://locust.io/`

[4] https://aag-it.com/the-latest-cloud-computing-statistics/

[5] `https://www.cisco.com/c/en/us/solutions/cloud/what-is-container-orchestration.html`

[6] `https://www.ibm.com/topics/container-orchestration`

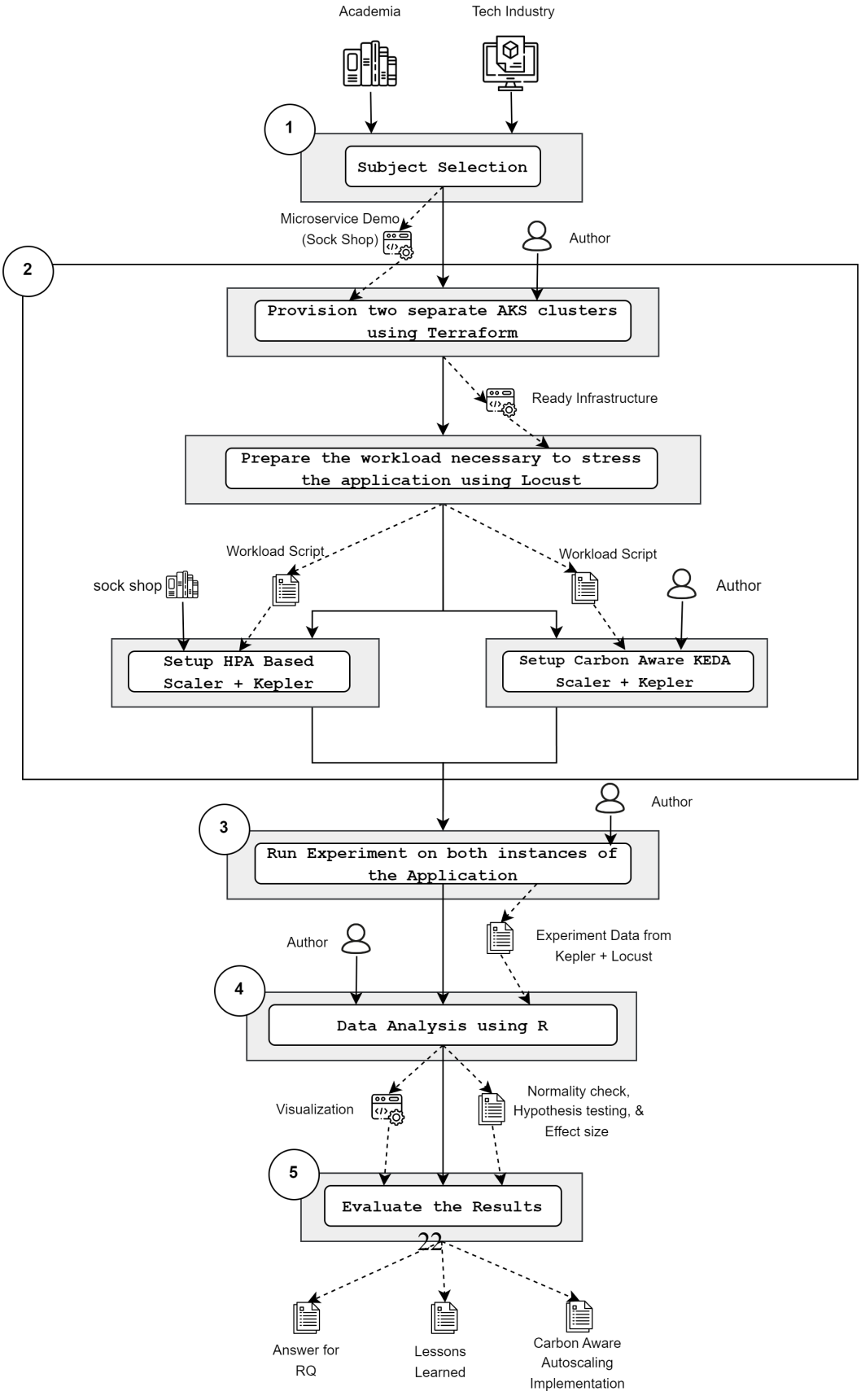


Figure 5: Visual Representation of the Study Design

veloper community support. We are specifically using managed Kubernetes service by Azure Kubernetes Services (AKS).

In **Step (2)** we set up the microservice application in public cloud, for this purpose we used Azure cloud provider, and set up two Azure Kubernetes Clusters (AKS)[7] on identical type of virtual machines using Terraform[8]. And finally we setup the two AKS clusters with the different Autoscaling strategy. For the Carbon Aware KEDA autoscaling we install Kubernetes Event-Driven Autoscaling (KEDA)[9], a component that provides event driven scale for any container running in Kubernetes, and deploy Carbon Aware KEDA Operator, an operator that helps KEDA scale Kubernetes workloads based on carbon intensity and Kubernetes Carbon Intensity Exporter[10] operator, which builds on the carbon-aware-sdk [11], to provide carbon intensity data in the Kubernetes cluster.

In order to make horizontal scaling carbon aware, we need to incorporate a mechanism that deals with the change in behaviour we want in response to the carbon intensity value of the energy it is consuming. Figure 6 provides an overview of the carbon-aware KEDA autoscaler's design, featuring essential components such as the Carbon Aware Software Development Kit (SDK), which acquires current and forecasted carbon intensity data from the public cloud infrastructure. This information is then utilized by the Carbon Aware KEDA operator. Additionally, Grafana is employed for visualizing key metrics, KEDA executes events, Prometheus collects relevant application metrics, and the microservice-based application is integrated. For a more in-depth understanding of the Carbon Aware KEDA operator's operation, refer to Figure 7.

Figure 7 illustrates the Carbon Aware KEDA operator's capability to adapt its scaling behavior dynamically using carbon intensity data from third-party sources like WattTime[12], or other providers. Importantly, this operator does not necessitate any changes in application or workload code and is compatible with any KEDA scaler. The carbon aware

[7] https://learn.microsoft.com/en-us/azure/aks/
[8] https://www.terraform.io/
[9] https://keda.sh/
[10] https://github.com/Azure/kubernetes-carbon-intensity-exporter/
[11] https://github.com/Green-Software-Foundation/carbon-aware-sdk
[12] https://www.watttime.org/

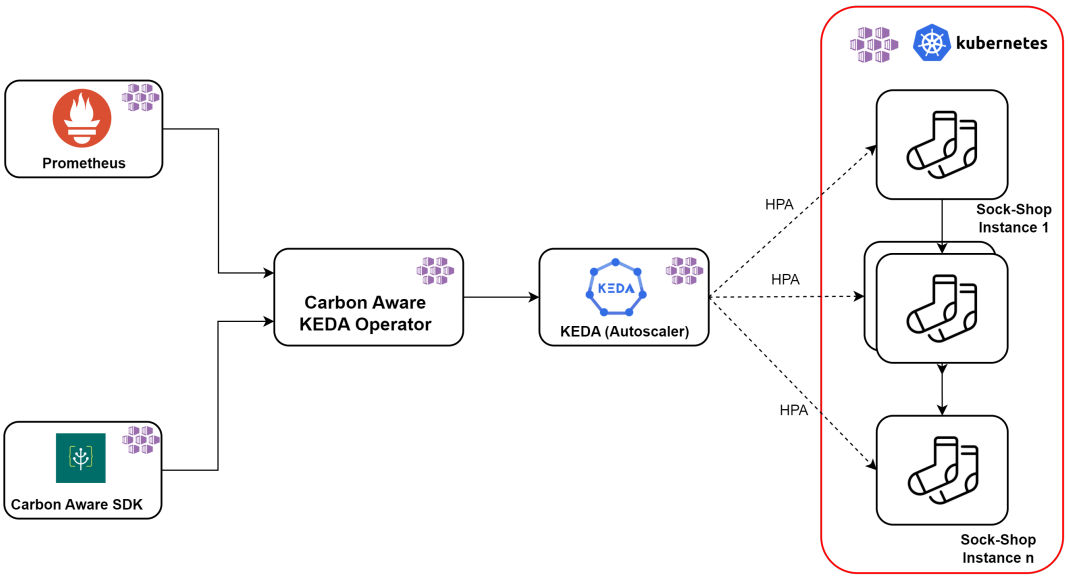


Figure 6: Visual Representation of the Carbon Aware KEDA Autoscaling for Sock Shop microservice-based application

KEDA operator retrieves carbon intensity data from a ConfigMap, generated by a third-party component. In our experiment, we utilized the Kubernetes Carbon Intensity Exporter Operator[13], leveraging the carbon-aware-sdk, to provide carbon intensity data within the Kubernetes cluster for carbon-aware decision making. The "Kubernetes carbon intensity exporter" retrieves 24-hour carbon intensity forecast data every 12 hours from WattTime third party grid carbon intensity data provider, updating the configmap accordingly. The current scaling logic based on carbon intensity alone remains independent of workload usage.

To set the thresholds, the idea is to find ranges between minimum and maximum carbon intensity and divide them into "buckets", based on the demo provided in the official documentation. For our experiment, we use 3 thresholds that represent "low", "medium", and "high" buckets where :

- The 3 buckets size is defined by: $\max - \min/3 = (400 - 150)/3 = 83.3$

[13] `https://github.com/Azure/kubernetes-carbon-intensity-exporter`

- Low bucket: carbon intensity is ≤ 233 ($\leq 120 + 83$)
- Medium bucket: carbon intensity is > 203 and ≤ 316 ($\leq 120 + 83 + 83$)
- High bucket: carbon intensity is > 286 and ≤ 400 (or higher > 400, since this is the highest threshold defined in the array)

Configuring thresholds in an array like this gives you flexibility to create as many thresholds/buckets as needed.

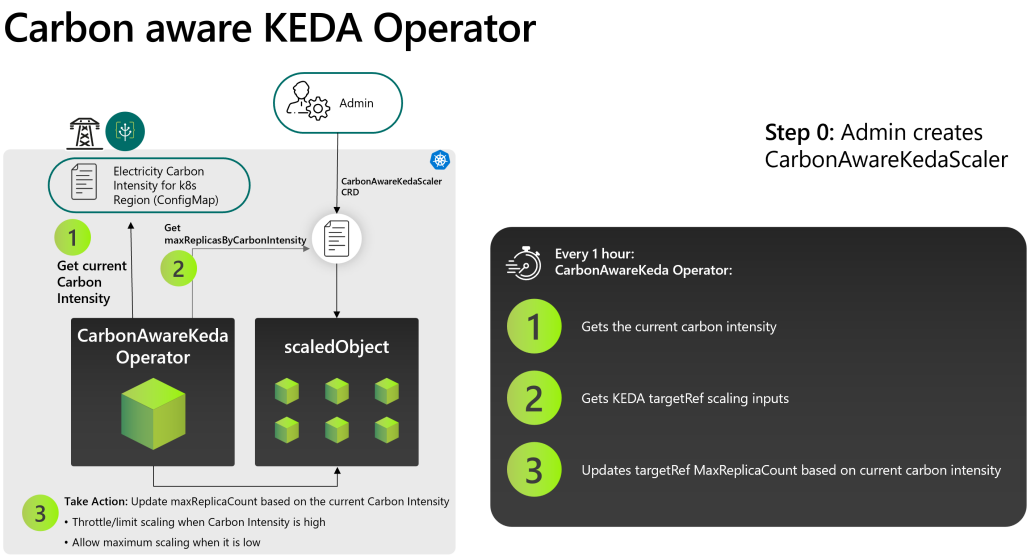
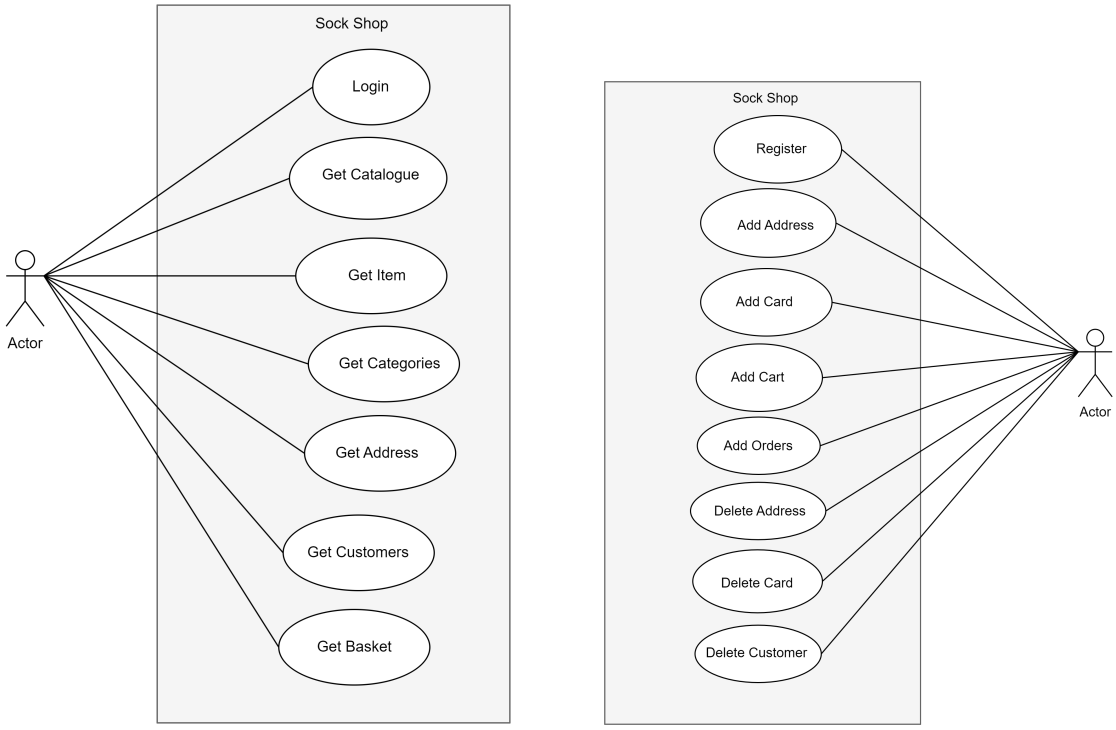


Figure 7: Comprehensive Visualization of the Carbon Aware KEDA Operator from the official open-source documentation on GitHub by Microsoft Azure[14].[57]

In **Step (3)**: We prepare workloads that simulate actual end user usage of the application, which are used to validate the overall functionality and can also be used to put simulated load on the system using Locust[15].

we run the experiment on both instances of the microservice based application, with the identified workload scenario. As mentioned above Sock Shop is a demo implementation of an ecommerce application in a microservice architectural style. As we were investigating

[15] `https://locust.io/`



(a) Read-Intensive Workload Scenario (b) Write-Intensive Workload Scenario

Figure 8: Use Case visualization of of both Read intensive and write intensive workload scenarios

best test case scenarios to generate workload, we considered two aspects. one the type of the application which is an ecommerce website and The API endpoint provided in the official documentation of the Sock shop. Based on these, we identified two test scenarios to generate workload for the microservice based application. These are Read Intensive workload Scenario and Write intensive workload scenario (Figure 8). We particularly think these workload scenarios are ideal for the task, as they mimic realistic simulation of e-commerce websites, as they often experience varying degrees of read and write operations due to customer interactions. By using these scenarios, we can more accurately simulate real-world user behaviours, such as browsing products(read-intensive) and placing orders(write intensive), they also provide comprehensive testing that covers critical aspects of the sock-shop, allowing us to assess the system's performance under different types of user interactions,

it also suits the scalability assessment of the sock shop by gauging the system's scalability by measuring its ability to handle increased loads and different system microservices is affected by read and write intensive scenarios which is what we want in for our experiment. For this experiment we used 300 virtual users and 30 Spawn rate to generate the needed workload. Detail description of this process is given in section 4.

In **Step (4)**, we performed data analysis using R scripts, which included data exploration, statistical description of the independent variable for each hypothesis, normality checks, data normalization, one-way Analysis of Variance (ANOVA), Welch two-sample t-test, Cohen's d effect size test, correlation tests, and necessary visualizations. Detail description of this process is given in section 6.

In **Step (5)**, we conduct a thorough analysis of the results to identify and communicate the key findings of the study. This is achieved through proper visualization and description of the data. The practical review concludes by summarizing the main takeaways from the analysis. Detail description of this process is given in section 7.

4 Experiment Planning

With the primary aim of exploring the energy consumption of carbon-aware autoscaling for microservice applications within the realm of the public cloud, we have meticulously devised a well-structured and reproducible experiment. The experimental design has been thoughtfully crafted to ensure that our objectives are pursued in a systematic manner, enabling reliable replication of the study.

4.1 Subject Selection

To ensure an appropriate subject selection for our experiment, we evaluated several microservice-based applications. After careful consideration, we opted to focus on the sock-shop microservice exemplar[1]. This particular application serves as a widely utilized resource in both industry and academia, aiding in the demonstration and testing of microservice and cloud-native technologies. The sock-shop application boasts a design that encompasses minimal expectations, utilizing Domain Name Service (DNS) for service discovery(Table 1). Furthermore, its compatibility with deployment on a Kubernetes cluster makes it well-suited for integration with Kepler. This integration enables us to capture and analyze various energy consumption metrics pertaining to different Kubernetes components, including Pods and Nodes.

During the initial phase of our study involving public cloud service providers, we conducted an extensive evaluation of the top 10 largest cloud providers, as listed in the technologymegazine[2]. Among these providers, we specifically selected Azure due to its robust offerings. Azure encompasses a comprehensive portfolio of over 200 products and cloud services, encompassing four distinct categories of cloud computing: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Software as a Service (SaaS), and Serverless. Additionally, Azure's reputation and technical expertise, particularly with our esteemed

[1] `https://microservices-demo.github.io/`

[2] `https://technologymagazine.com/top10/top-10-biggest-cloud-providers-in-the-world-in-2023`

thesis advisor specializing in this domain, led us to choose Azure as the public cloud infrastructure for our experiment.

Component	*Language / Framework(Port)*	**Database / Broker(Port)**
Carts	Java / Spring Boot (80)	MongoDB (27017)
Catalogue	Go (80)	MySQL (3306)
Frontend	HTML + Javascript (80)	Redis (6379)
Orders	Java / Spring Boot (80)	MongoDB (27017)
Payment	Go (80)	
Queue Master	Java / Spring Boot (80)	RabbitMQ (5672)
Shipping	Java / Spring Boot (80)	RabbitMQ (5672)
User	Go (80)	MongoDB (27017)

Table 1: Overview of Sock Shop Microservices Components, Covering Various Use Cases from Frontend to Persistence and their technology stack

4.2 Experiment Variable

In order to address the research questions outlined in Section 3.2, our study focuses on Autoscaling strategies as the primary independent variable. As a result, we have identified two key treatments derived from the autoscaling approach:

1. Carbon-Aware KEDA autoscaling strategy
2. HPA-Based autoscaling strategy

The experiment involves the utilization of a microservice application served with either the Carbon-Aware KEDA Autoscaling strategy, which incorporates the Carbon Intensity Exporter through Kubernetes Event Driven Autoscaling (KEDA)[3], or the HPA-Based autoscaling strategy. To ensure consistency across both treatments, we employ Kepler, Prometheus[4], and Grafana[5] as common components while deploying the microservice application on the

[3] `https://keda.sh/`
[4] `https://prometheus.io/docs/introduction/overview/`
[5] `https://grafana.com/`

Azure platform. To enhance the experiment's reliability, we leverage Terraform[6] for identical cloud resource provisioning, enabling us to create a standardized infrastructure for deploying the microservice application.

Name	Description
Energy consumption	The energy consumption, measured in joules (J), refers to the amount of energy utilized by the public cloud to execute the workloads across various microservices' pods.
Response Time	The response time, measured in milliseconds (ms), denotes the duration taken by the microservice application server to handle requests generated by the workload.

Table 2: Dependent variables considered for this research

4.3 Experimental Hypothesis

Based on the experiment definition, the mean value **(μ)** of **{**vwc**}** will be used to calculate the difference for each variable. v represents the dependent variables **{**e**,** r**}** described in Table 3. w indicates different workloads, while c represents a subset of enabled carbon-aware settings for a microservice application **{**$true$**,** $false$**}**. The nature of the experiment is a **two-tailed hypothesis**.

Let μewm represent the average value of a dependent variable(*e*) when running different workloads(*w*), and a subset of microservices configuration (*m*), The null and alternative hypotheses for the research questions are formulated as follows:

For microservices configuration:

$$H_{0,wm} : \mu_{ewm} = \mu_{ewm}, \forall w \wedge \forall m$$
$$H_{a,b} : \mu_{ew\boldsymbol{false}} \neq \mu_{ew\boldsymbol{true}}, \forall w$$

The null hypothesis *(H0,wc)* posits that there is no significant difference in the average energy consumption values of the Carbon Aware KEDA components for different configurations of microservices. Conversely, the alternative hypothesis *(Ha,b)* suggests that

[6] `https://www.terraform.io/`

there is a significant difference in in the average energy consumption values of the Carbon Aware KEDA components for different configurations of microservices.

Given that μewc represents the average value of a dependent variable (e) when running different workloads(w), and a subset of enabled carbon-aware settings (c), the null and alternative hypotheses for the research questions is formulated as follows:

For energy consumption:

$$H_{0,wc} : \mu_{ewc} = \mu_{ewc}, \forall w \wedge \forall c$$
$$H_{a,b} : \mu_{ew\boldsymbol{false}} \neq \mu_{ew\boldsymbol{true}}, \forall w$$

The null hypotheses (*H0,wc*) states that, there is no significant difference in the average energy consumption values of the microservice-based application between Carbon Aware KEDA and HPA Based Autoscaling methods. Conversely, the alternative hypotheses *(Ha,b)* states that there is significant difference in the average energy consumption values of the microservice-based application between Carbon Aware KEDA and HPA Based Autoscaling methods.

The null and alternative hypotheses for the research questions are formulated based on the average response time value of the dependent variable *(r)*, denoted by μrwc, while considering different workloads *(w)* and a subset of enabled carbon-aware settings *(c)*.

For Response Time:

$$H_{0,wc} : \mu_{rwc} = \mu_{rwc}, \forall w \wedge \forall c$$
$$H_{a,b} : \mu_{rw\boldsymbol{false}} \neq \mu_{rw\boldsymbol{true}}, \forall w$$

The null hypothesis *(H0,wc)* suggests that there is no significant difference in the average response time values of the microservice-based application between Carbon Aware KEDA and HPA Based Autoscaling methods. Conversely, the alternative hypothesis *(Ha,b)* proposes that there is a significant difference in the average response time values of the microservice-based application between Carbon Aware KEDA and HPA Based Autoscaling methods.

Similarly for the correlation, where ρ represents the population correlation coefficient, the null and alternative hypotheses for the research questions is formulated as follows:

For Energy and Response Time Correlatoin:

$$H_1 : \rho = 0$$
$$H_a : \rho \neq 0$$

The null hypothesis *(H1)* posits that there is no statistically significant correlation between the energy consumption and response time of the microservice-based application when operating on either of the autoscalers. Conversely, the alternative hypothesis *(Ha)* suggests that there is a statistically significant correlation between the energy consumption and response time of the microservice-based application when operating on either of the autoscalers.

4.4 Experiment Design

We conducted a rigorous 2x2 factorial design experiment to assess the influence of different autoscaling strategies on the dependent variables (energy consumption and response time) in a microservice application. We thoroughly measured the dependent variables for each combination of autoscaler, workload scenarios, and hypotheses, ensuring comprehensive analysis of the impacts.

To address RQ1, we conducted four configurations of microservices (1 MS, 4 MS, 8 MS, and 14 MS), with measurements taken both with and without workload to assess the energy overhead imposed by Carbon Aware KEDA components. To ensure accuracy and account for variability[58], each trial was repeated six times for every microservice configuration. For RQ2 and RQ3, we tested the microservice-based application with all combinations of autoscaler and workload scenarios. To mitigate bias and account for computational resources, the application was cleared and redeployed after each trial. To enhance precision, each trial was measured six times, focusing solely on dependent variables during workload processing.

4.5 Data Analysis

The collected data consists of numerical values for each dependent variable under different workload scenarios with both autoscaler treatments. This data is subjected to quantitative

analysis, and the decision process for its analysis is illustrated in figure 9, with a detailed explanation provided below.

Data Exploration: The collected data is initially subjected to descriptive statistics, calculating key metrics such as the minimum value, first quartile, median, mean, third quartile, maximum value, and standard deviation for each variable, namely, energy consumption and response time. The data is then visualized using histograms to gain insights into the distribution and dispersion of each metric.

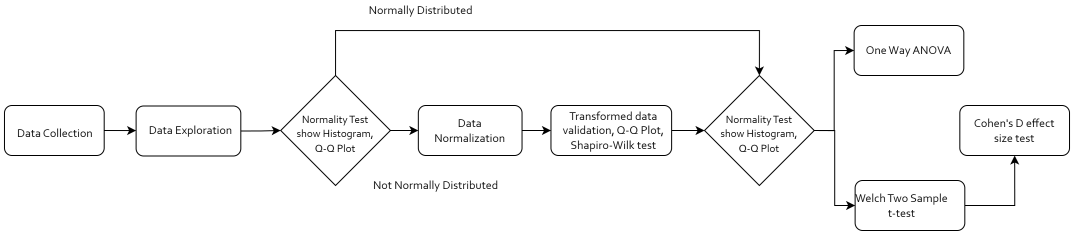


Figure 9: Visual Representation of the Analysis Decision

Normality Check: Subsequently, a Shapiro-Wilk test[7] is conducted to assess whether the data approximates a normal distribution. The resulting p-value from the test indicates the likelihood of the data being normally distributed. If the p-value is less than 0.05, it suggests that the data is not normally distributed. Conversely, if the p-value is greater than or equal to 0.05, it indicates that the data can be assumed to follow a normal distribution.

Non-Normal Data Transformation: If the data is found to be not normally distributed, the bestNormalize R package is utilized to transform it into a normalized form. Specifically, the orderNorm[8] transformation option is selected for this experiment. The transformation process is visualized using a Q-Q plot, where the data should approximate a diagonal line if it is normally distributed. The Shapiro-Wilk normality test is then performed on the transformed data to verify normality. If the p-value from this test is greater than 0.05, it indicates that the data can be considered normally distributed.

[7] `https://en.wikipedia.org/wiki/Shapiro%E2%80%93Wilk_test`

[8] `https://www.rdocumentation.org/packages/bestNormalize/versions/1.9.0/topics/orderNorm`

Normality Assumption: For RQ1, since there are four different treatments representing the number of microservice configurations (01 MS, 04 MS, 08 MS, and 14 MS), a one-way ANOVA is employed to assess the statistical significance of differences among these populations. On the other hand, the Welch two-sample t-test is utilized to test the significance between the populations of energy consumption and response time for the two instances of autoscaler treatments. As the Welch two-sample t-test only shows the overall significant difference in result. Cohen's D test is done to show the effect size of the result.

4.6 Study Replicability

To facilitate experiment verification and replication, we publicly share the replication package on GitHub[9]. The package includes (i) YAML scripts for microservice application deployment, HPA scalers, KEDA Scalers, and Carbon Aware KEDA Scalers, (ii) raw data from the experiment, (iii) R scripts for data analysis, (iv) relevant diagrams not included in the paper, and (v) a README file providing guidelines for replicating the experiment.

[9] `https://github.com/HabsB/Green-Autoscaling`

5 Experiment Execution

This section presents the technical setup employed for conducting the experiment. It covers the tools, infrastructure, and software used for deploying microservices and measurements. The experiment execution is divided into three main steps: (1) experiment preparation, (2) infrastructure setup, and (3) experiment execution and measurement. Each step is detailed in the following paragraphs.

5.1 Experiment Preparation

The experiment utilized the Azure public cloud with virtual machines of type Standard_D2s_v5, as detailed in Table 3. All experimental trials were conducted on this infrastructure. Energy consumption of the microservices was measured using Kepler. Prometheus was employed for metric recording and Grafana for visualization. Remote code execution was facilitated through the Azure Command Line Interface (CLI). The Azure server operated on Common Base Linux Mariner, VERSION=”2.0.20230621”.

The experiment preparation begins by installing Azure AKS on both application instances. The selected subject’s microservice-based application, along with all its microservices, is then deployed using the kubectl command. After ensuring that the application is running, the scalers are deployed in a similar manner. For the Carbon Aware KEDA instance, a custom resource called ”CarbonAwareKedaScaler” is also deployed to set the maximum replicas that KEDA can scale up to based on carbon intensity. Figure 4 presents a schematic overview of the experiment infrastructure, comprising two components: (1) an Azure infrastructure responsible for deploying the subject and all carbon-aware KEDA components, as well as collecting experimental data, and (2) a client-side from a laptop that initiates, concludes, and visualizes the test results of the experiment.

Size	Standard_D2s_v5
vCPU	2
Memory: GiB	8
Temp storage (SSD) GiB	Remote Storage Only
Max data disks	4
Max uncached disk throughput: IOPS/MBps	3750/85
Max burst uncached disk throughput: IOPS/MBps[2]	10000/1200
Max NICs	2
Max network bandwidth (Mbps)	12500

Table 3: Virtual machine specification used for the Azure infrastructure

5.2 Infrastructure Setup

To control the experiment environment, we installed Microsoft Azure CLI[1], a cross-platform command-line tool, on the laptop. The deployment of the microservice-based application, KEDA scalers, and custom resources was prepared in Yet Another Mark-up Language (YAML)[2] files to be executed by Azure CLI. Workload scenarios were managed using Locust, a scriptable and scalable performance testing tool written in Python. Prometheus, Grafana, Kepler, and all custom scraping resources were preinstalled on both servers. Additionally, KEDA, Carbon Aware KEDA operator, and the Carbon Intensity Exporter were preinstalled on the AKS cluster to leverage the Carbon Aware KEDA autoscaling strategy in one of the servers.

First, all deployments and services of the Sock Shop microservices-based application are installed in the ”sock-shop” namespace, as shown in Step 1a of Figure 10. Next, the appropriate Scaler script, either HPA or Carbon Aware KEDA, is applied depending on the instance of the application, as shown in Step 1b. In the case of Carbon Aware KEDA autoscaling, custom scalers are applied (Step 1c). Once all deployments are ready, the workload scenarios are executed using Locus in Step 2. Energy consumption data is collected

[1] `https://github.com/Azure/azure-cli`
[2] `https://yaml.org/`

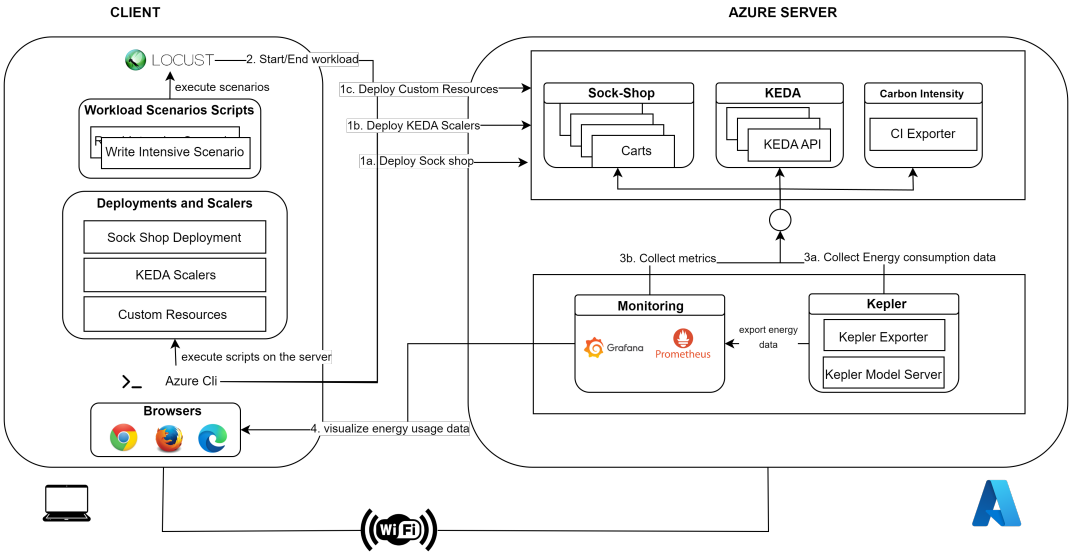


Figure 10: Visual Representation of Experiment Setup

using Kepler, which leverages extended Berkeley Packet Filter (ePBF) to probe CPU performance counters and Linux kernel tracepoints. This data is then exported to Prometheus along with other performance metrics (Step 3). Energy consumption can be visualized in Grafana using the ”port-forward” command, accessible through any browser (Step 4). Each treatment and workload scenario combination is repeated 6 times, and between each run, all deployments are removed, and resources are released to ensure a clean environment for each iteration.

5.3 Measurement

During each run of the experiment, measurements of energy consumption and response time are collected and stored. To obtain the energy consumption, the average energy consumed by each replica during the workload’s time stamp is calculated, and the sum of the averages for all replicas in the pod is determined. The average response time is measured using Locust after each run. Data for each treatment is then aggregated. To ensure accurate measurement and data collection, certain conditions were implemented in the execution environment. To avoid biases caused by network instability, all other browsers were closed,

and the only device connected to the WiFi was the laptop conducting the experiment. Each trial was performed separately to prevent caching or unreleased resources from affecting the results. The experiment environment was kept clean after each trial, and all resources were redeployed for the next round of the experiment.

6 Evaluation

In this section, we present the experiment results, exploring the collected data to gain insights into its distribution. Subsequently, the data is assessed for normality, and the hypotheses defined in Section 4 are tested. The outcomes concerning energy consumption and response time are displayed through appropriate plots and tables.

6.1 Carbon Aware KEDA components and their energy overhead (RQ1)

Data Exploration: Table 4 displays descriptive statistics for energy consumption of the Carbon Aware Keda components with varying microservices configurations (one, four, eight, and fourteen microservices). The mean energy consumption remains consistent, ranging from approximately **8.86 to 8.91**, as the number of microservices increases. The standard deviations (SD) of energy consumption across different configurations vary slightly, ranging from approximately **1.873 to 1.950**, indicating some variability in energy consumption among setups while remaining relatively close to the mean. To provide an overview of energy consumption based on the number of microservices, a box plot is used (Figure 11).

	1 Microservice	**4 Microservices**	**8 Microservices**	**14 Microservices**
Minimum	6.41	6.347	6.552	6.368
1^{st} Quartile	7.212	7.278	7.322	7.233
Median	9.619	9.615	9.614	9.614
Mean	8.866	8.867	8.911	8.86
Standard Deviation	1.949	1.941	1.873	1.95
3^{rd} Quartile	10.327	10.327	10.325	10.326
Maximum	10.573	10.564	10.563	10.563

Table 4: Descriptive Statistics for RQ1 - Number of Microservices and their Energy overhead

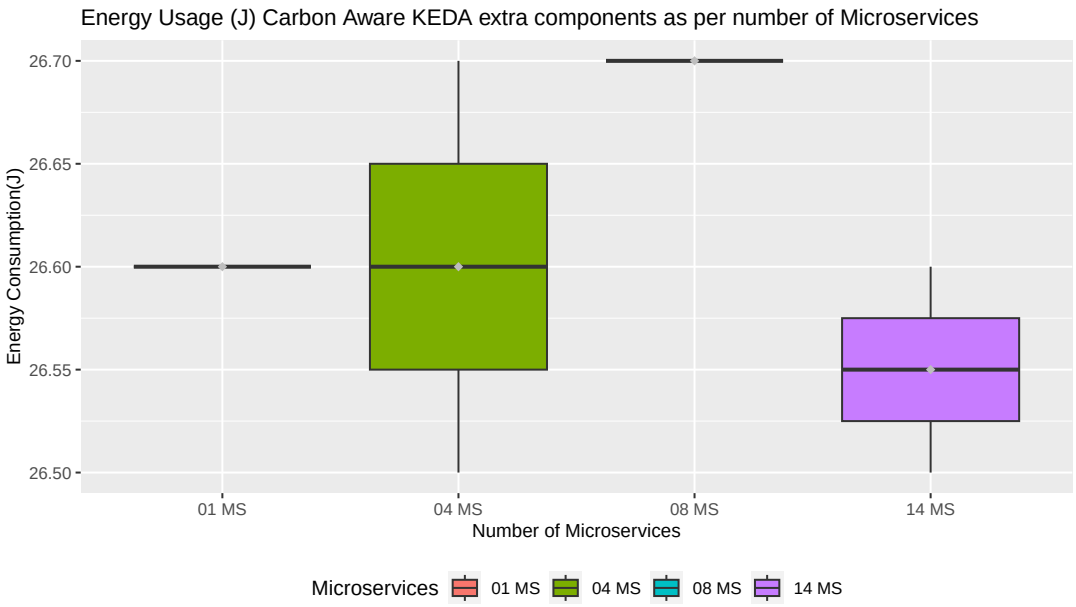


Figure 11: Box Plot visualization of energy consumption overhead by Carbon Aware KEDA components as per the number of Microservices

Normality Check: Before testing the hypothesis, it is crucial to assess normality, as it determines the choice between parametric or non-parametric tests and influences measures of central tendency and dispersion[59]. Figure 12 displays a histogram of the energy usage distribution obtained from the microservices experiment, indicating non-normality. To confirm this, the Shapiro-Wilk normality test[60] was conducted, resulting in $W = 0.73858$ and a $p\text{-value} < 3.408e\text{-}05$, confirming non-normality, as the p-value is below the significance threshold of 0.05.

Prior to conducting the ANOVA test[61], the data undergoes orderNorm[1] transformation to achieve normal distribution, as demonstrated in the Q-Q plot[2] (Figure 13. The normality is further confirmed by the Shapiro-Wilk test for the transformed data, with $W = 0.99605$ and $p\text{-value} = 1$.

[1] `https://rdrr.io/github/petersonR/bestNormalize/man/orderNorm.html`
[2] `https://it.wikipedia.org/wiki/Q-Q_plot`

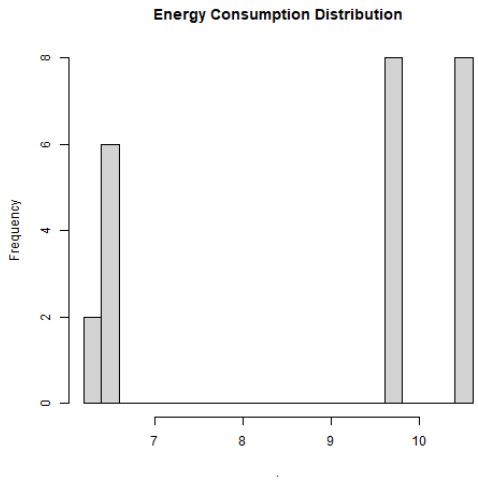


Figure 12: Histogram visualization for distribution of energy consumption overhead by Carbon Aware KEDA components as per the number of Microservices

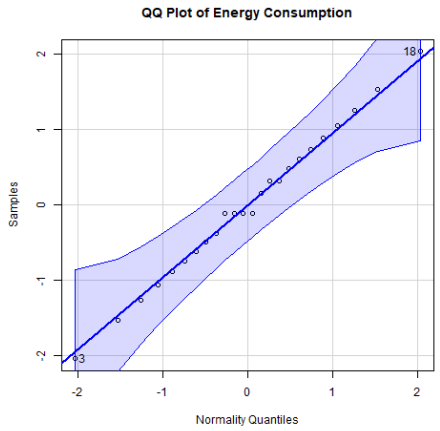


Figure 13: Q-Q Plot visualization for normalized data of energy consumption overhead by Carbon Aware KEDA components as per the number of Microservices

Hypotheses Testing: *H01: The null hypothesis posits that the average energy consumption values for the Carbon Aware Keda components with varying numbers of microservices are equal.* To test this hypothesis, a one-way ANOVA test is conducted on the components, considering four treatments. As shown in Table 5, the ***p-value*** exceeds the significance level of ***0.05***, leading to the ***failure to reject the null hypothesis***.

	Df	Sum Sq	Mean Sq	F value	p-value
Microservices	1	0.384	0.3844	0.379	0.544
Residuals	22	22.318	1.0145		

Table 5: One way ANOVA for RQ1 - Number of Microservices Vs Energy Overhead

6.2 Energy comparison of HPA Based & Carbon Aware KEDA Autoscaling (RQ2)

Data Exploration: Table 6 presents the results of the data exploration for the energy consumption of the two Autoscalers - Carbon Aware KEDA and HPA Based as per microservices. Each method comprises 336 data points, providing a robust sample size for analysis. The descriptive statistics offer valuable insights into the distribution of energy consumption within each method. For the Carbon Aware KEDA Autoscaler, the energy consumption ranges from 3.11 to 29.94, with a mean value of 9.66. The standard deviation, which measures the spread of data points around the mean, is calculated to be 7.266. This indicates a moderate level of variability in energy consumption across the data points. In comparison, the HPA Based method exhibits a wider range of energy consumption, spanning from 3.13 to 65.58. The mean energy consumption for this autoscaler is higher at 15.642, and the standard deviation is larger, measuring at 16.079. These results suggest a greater variation in energy consumption within the HPA Based method, with some data points experiencing significantly higher energy usage compared to others. To present a comprehensive view of energy consumption relative to the autoscaler employed, a box plot is employed (Figure 14). Similarly, The statistical summary table 7 compares two autoscaling approaches, ”Carbon Aware KEDA” and ”HPA Based”. Carbon Aware KEDA generally demonstrated

lower values for key metrics such as minimum, median, mean, and maximum, suggesting its potential efficiency and effectiveness compared to HPA$_{B}ased$.

	Carbon Aware KEDA	**HPA Based**
Minimum	3.11	3.13
1st Quartile	4.252	3.245
Median	6.59	7.17
Mean	9.66	15.642
Standard Deviation	7.266	16.079
3rd Quartile	13.67	28.422
Maximum	29.94	65.58

Table 6: Descriptive Statistics for RQ2 - Energy Consumption comparison of HPA based and Carbon Aware KEDA Autoscalers as per each microservices

Autoscaling	Carbon_Aware_KEDA	HPA_Based
Minimum	104.0	175.01
1st Quartile	114.0	181.0
Median	115.0	221.0
Mean	125.01	219.01
Standard Deviation	21.1	38.6
3rd Quartile	138.	256.
Maximum	164.0	260.0

Table 7: Descriptive Statistics for RQ2 - Energy Consumption comparison of HPA based and Carbon Aware KEDA Autoscalers for microservice based application

Normality Check: The energy consumption data obtained from the experiment was assessed for normality using both a histogram and the Shapiro-Wilk test. The histogram displayed in Figure 15 revealed that the data is not normally distributed. The Shapiro-Wilk test further supported this finding, yielding a significant p-value of **9.157e-16**, well below the significance threshold of **0.05**.

To enable t-test analysis, the data was transformed into a normal distribution using the orderNorm transformation. The effectiveness of the transformation was confirmed through the Q-Q plot in Figure 16, which displayed a visually normal distribution. Subsequently,

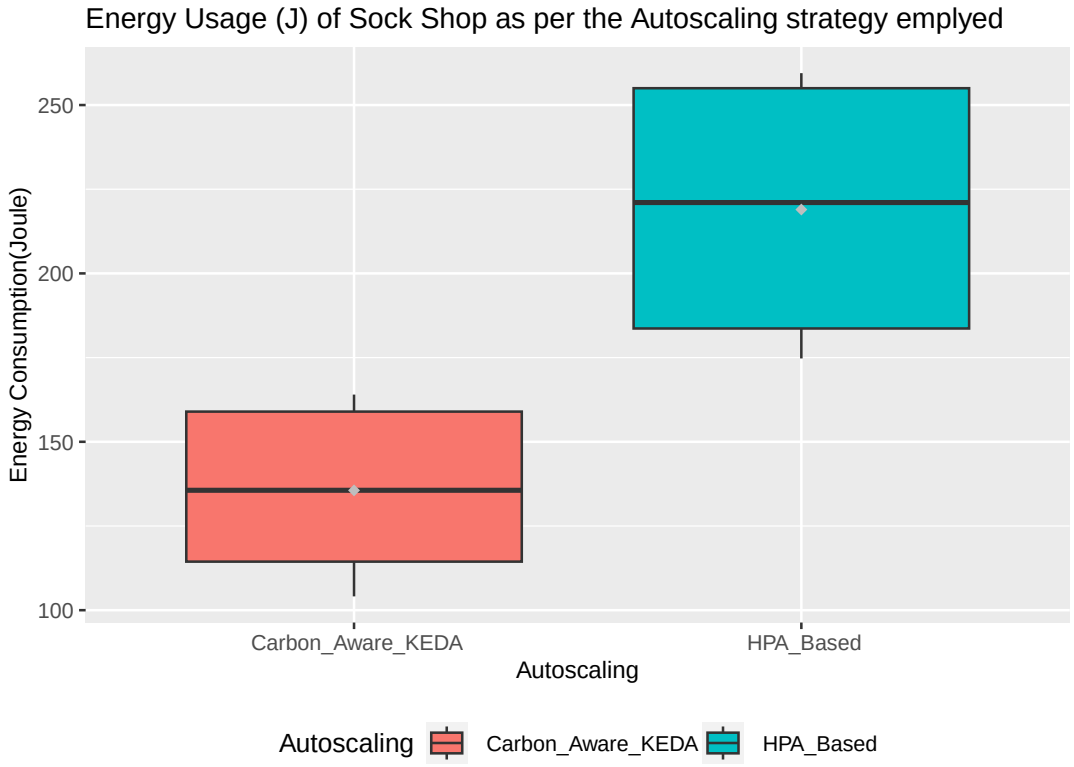


Figure 14: Box Plot visualization of energy consumption by the microservice based application as per the Autoscaling strategy employed

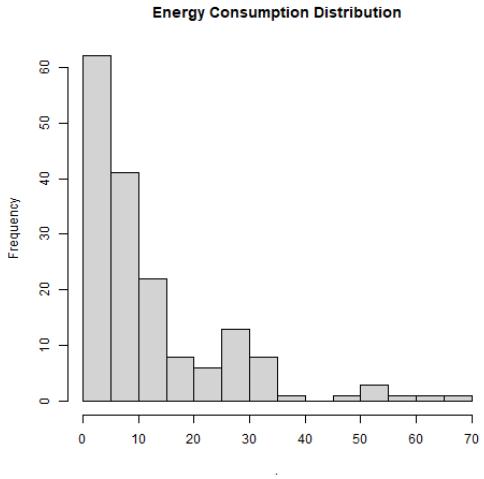


Figure 15: Histogram visualization for distribution of Energy Consumption of the microservice based application for both HPA based and Carbon Aware KEDA Autoscalers

the Shapiro-Wilk test for the transformed data yielded a high p-value of **0.9912**, further confirming its normality.

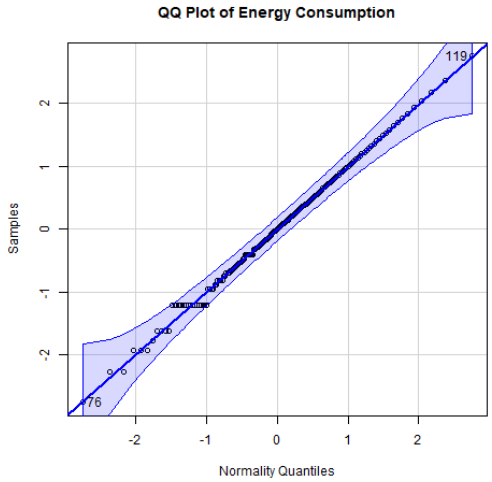


Figure 16: Q-Q Plot visualization for normalized data of Energy Consumption of the microservice based application for both HPA based and Carbon Aware KEDA Autoscalers

Hypotheses Testing: *H01: The null hypothesis states that there is no significant difference in the average energy consumption values of the microservice-based application between Carbon Aware KEDA and HPA Based Autoscaling methods.* A Welch two-sample t-test[62] was conducted to compare the average energy consumption values of the microservice-based application for Carbon Aware KEDA and HPA Based Autoscaling methods. The test resulted in a ***p-value of 0.002497***, which is less than the significance level of ***0.05***, indicating a significant difference between the two autoscalers. Therefore, we ***reject the null hypothesis*** and accept the *alternative hypothesis*, concluding that there is a significant difference in the average energy consumption values between the two methods.

Effect Size: To assess the magnitude of the alternative hypothesis, we examine the Cohen's d effect size[63], which yields a value of -2.42, indicating **a large effect size**. The 95% confidence interval for the effect size is [-3.93, -0.85]. This indicates a substantial difference in the average energy consumption of the microservice-based application between the two autoscalers being compared. The autoscaling strategy has a significant impact on the outcome, demonstrating its importance in influencing energy consumption.

The density curves of energy usage for both autoscalers is illustrated by Figure 21

6.3 Energy Saving impact on Response Time (RQ3)

Data Exploration: The statistical summary table 8 presents data for two autoscaling methods, Carbon Aware KEDA and HPA Based. For the Carbon Aware KEDA method, the response time ranges from 1740 to 2792 (ms), with a mean of 2206.433 and a standard deviation of 443.471. On the other hand, the HPA Based method shows a narrower response time range, from 1404.9 to 1854.2, with a lower mean of 1639.133 and a smaller standard deviation of 187.083. To provide a comprehensive perspective on response time concerning the autoscaler utilized, a box plot is employed for visualization (Figure 17).

Normality Check: Normality of the response time data obtained from the experiment was evaluated using a histogram and the Shapiro-Wilk test. While the histogram and Q-Q plot in Figure 20 did not clearly indicate the distribution's normality, the Shapiro-Wilk test

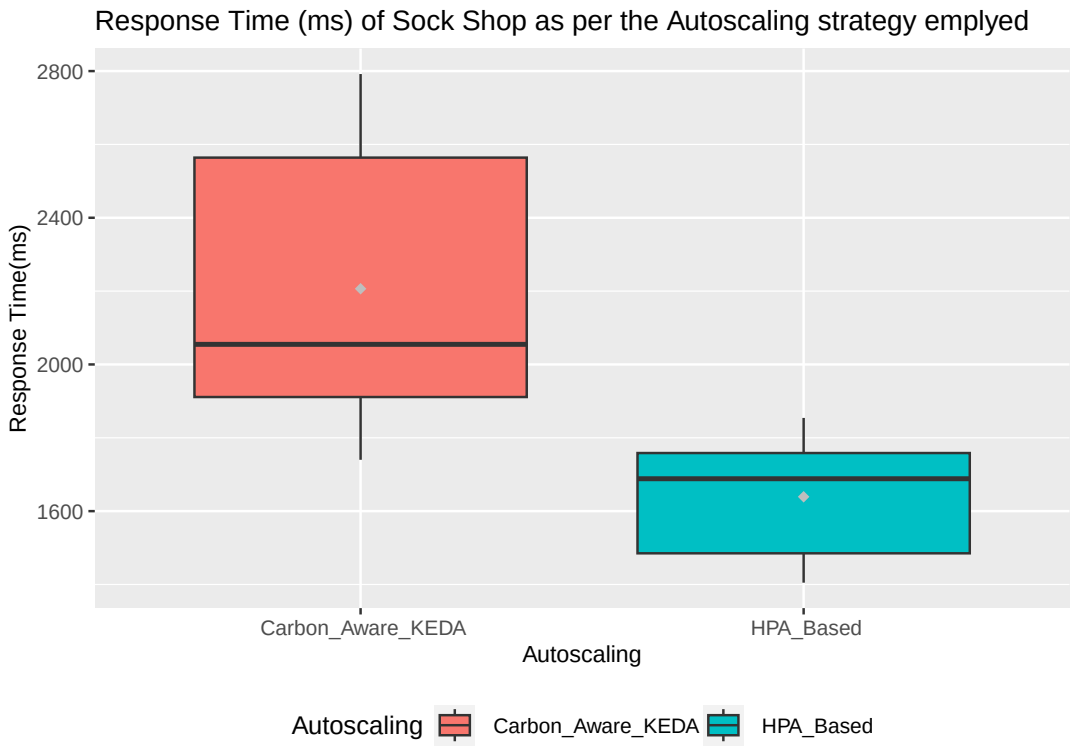


Figure 17: Box Plot visualization of response time by the microservice based application as per the Autoscaling strategy employed

yielded a p-value of **0.0551**, which is greater than the significance threshold of **0.05**. This result suggests that the data can be considered as normally distributed.

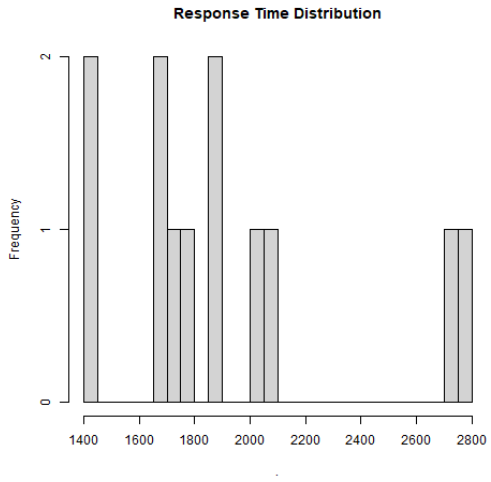


Figure 18: Histogram Visualization

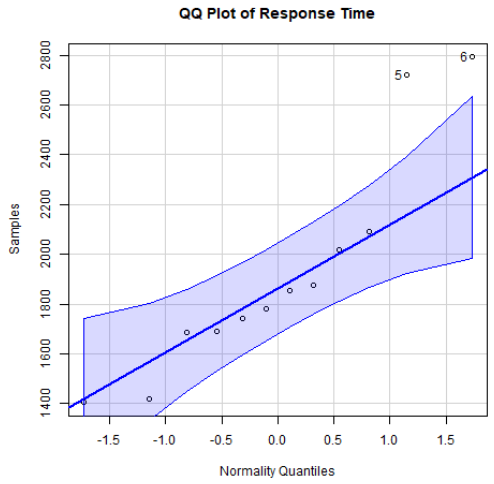


Figure 19: Q-Q Plot visualization

Figure 20: Histogram & Q-Q plot visualization of response time for both HPA based and Carbon Aware KEDA Autoscalers

	Carbon Aware KEDA	**HPA Based**
Minimum	1740	1404.9
1st Quartile	1911.125	1409.9
Median	2054.8	1485
Mean	2206.433	1688.35
Standard Deviation	443.471	187.083
3rd Quartile	2564.025	1758.425
Maximum	2792	1854.2

Table 8: Descriptive Statistics for RQ3 - Response time of both HPA based and Carbon Aware KEDA Autoscalers

Hypotheses Testing: *H01: The null hypothesis states that there is no significant difference in the average response time values of the microservice-based application between Carbon Aware KEDA and HPA Based Autoscaling methods.* A Welch two-sample t-test

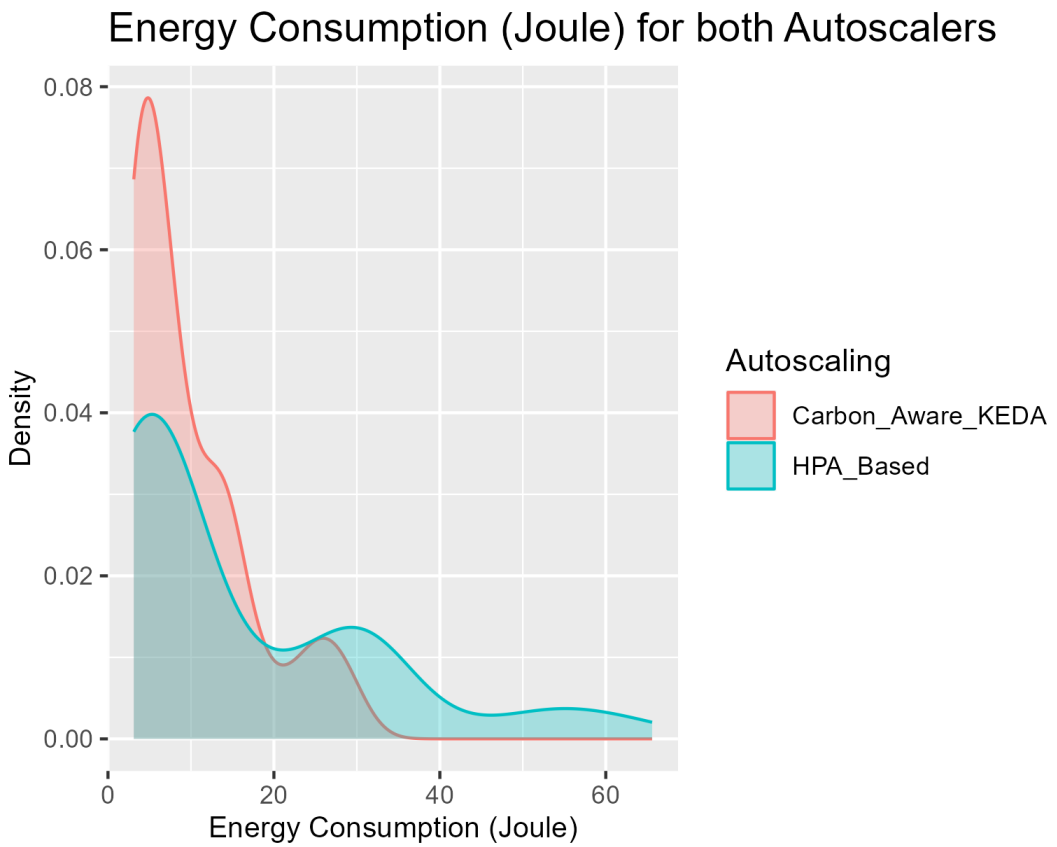


Figure 21: Density Curve visualization of energy usage for both HPA based and Carbon Aware KEDA Autoscalers

was performed to compare the average energy consumption values of the microservice-based application for Carbon Aware KEDA and HPA Based Autoscaling methods. The test yielded a p-value of **0.02447**, which is less than the significance level of **0.05**, indicating a significant difference between the two autoscalers. Consequently, we ***reject the null hypothesis***, which posited no significant difference, and accept the *alternative hypothesis*, concluding that there is a notable distinction in the average energy consumption values between the two methods.

Effect Size: The Cohen's d effect size of 1.67 denotes a **significant impact size**. The 95% confidence interval [0.30, 2.98] indicates a substantial difference in means between the response time of the microservice-based application for the two autoscalers. The density curves of energy usage for both autoscalers is illustrated by 22

Hypotheses Testing: *H02: The null hypothesis postulates that there is no statistically significant correlation between the energy consumption and response time of the microservice-based application when operating on either of the autoscalers.* The Pearson's product-moment correlation coefficient[64] for the relationship between Energy Consumption and Response Time variables is -0.6029, indicating a moderate negative correlation between the two. The statistically significant p-value of **0.03798** at a significance level of **0.05** supports the existence of this correlation. The 95% confidence interval for the correlation coefficient [-0.8743, -0.0443] further confirms the significant negative association between Energy Consumption and Response Time. In summary, the findings suggest that as the Energy consumption of the microservice-based application decreases, the response time of the application tends to increase, with **a moderate level of correlation** between the variables. These results provide evidence to ***reject the null hypothesis (H02)*** and indicate that there is indeed a correlation between energy consumption and response time when running on both the Carbon Aware KEDA and HPA Based autoscalers.

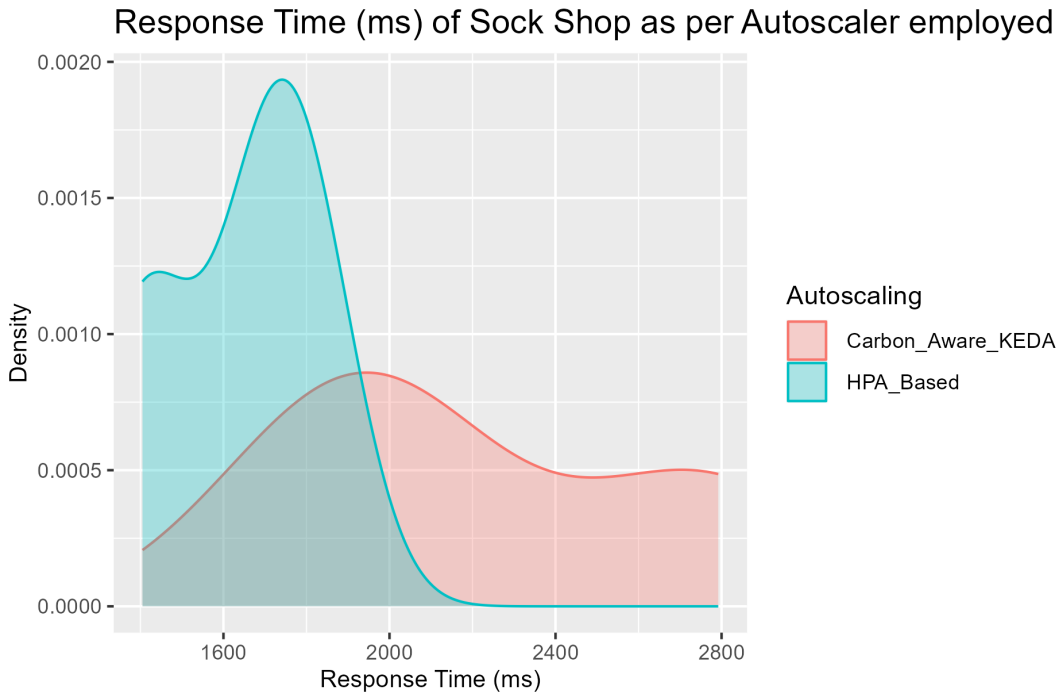


Figure 22: Density Curve visualization of response time for both HPA based and Carbon Aware KEDA Autoscalers

7 Discussion

This section delves into the implications and interpretations of the results discussed in Section 6, organized according to the respective research questions. Each interpretation is presented in detail to provide a comprehensive understanding of the findings.

7.1 Results and Findings

RQ1. What is the relationship between the number of microservices and the energy consumption overhead caused by carbon-aware KEDA autoscaling components in the public cloud?

To address RQ1, a hypothesis related to the number of microservices and its impact on energy consumption was tested using a one-way ANOVA test. The results led to the acceptance of the null hypothesis, indicating that the number of microservices does not have a statistically significant impact on the energy overhead imposed by the Carbon Aware KEDA components. This finding is further illustrated in Figure 23, considering both scenarios with and without applying workload.

The investigation of energy consumption cost associated with the Carbon Aware KEDA components yielded an intriguing result. The energy overhead remained consistent regardless of workload and number of microservices, as depicted in Figure 23, which suggests that the energy cost for running 4 microservices is similar to that of running 14 microservices. As a result, utilizing the Carbon Aware KEDA autoscaling becomes more advantageous when managing larger numbers of microservices, such as in scenarios with 10s and 100s of microservices.

Finding 1

The energy cost efficiency of the Carbon Aware KEDA autoscaling provides a significant advantage in managing larger quantities of microservices. The energy cost remains constant across different configurations, making it a practical and effective approach for optimizing energy consumption in extensive microservices deployments without incurring additional costs as the microservices scale.

RQ2: How does the Carbon-Aware KEDA Autoscaling compare to HPA-Based autoscaling strategies in terms of energy consumption?

The null hypothesis states that there is no significant difference in the average energy consumption values of the microservice-based application between Carbon Aware KEDA and HPA Based Autoscaling methods. The energy efficiency comparison of the microservice-based application under these two autoscaling methods resulted in rejecting the null hypothesis, indicating that the Carbon Aware KEDA autoscaler leads to significantly less energy consumption of the microservice-based application in comparison to the HPA based autoscaler. The difference was confirmed by the Cohen's d effect size test, which indicated that the Carbon Aware KEDA autoscaler significantly reduces energy consumption

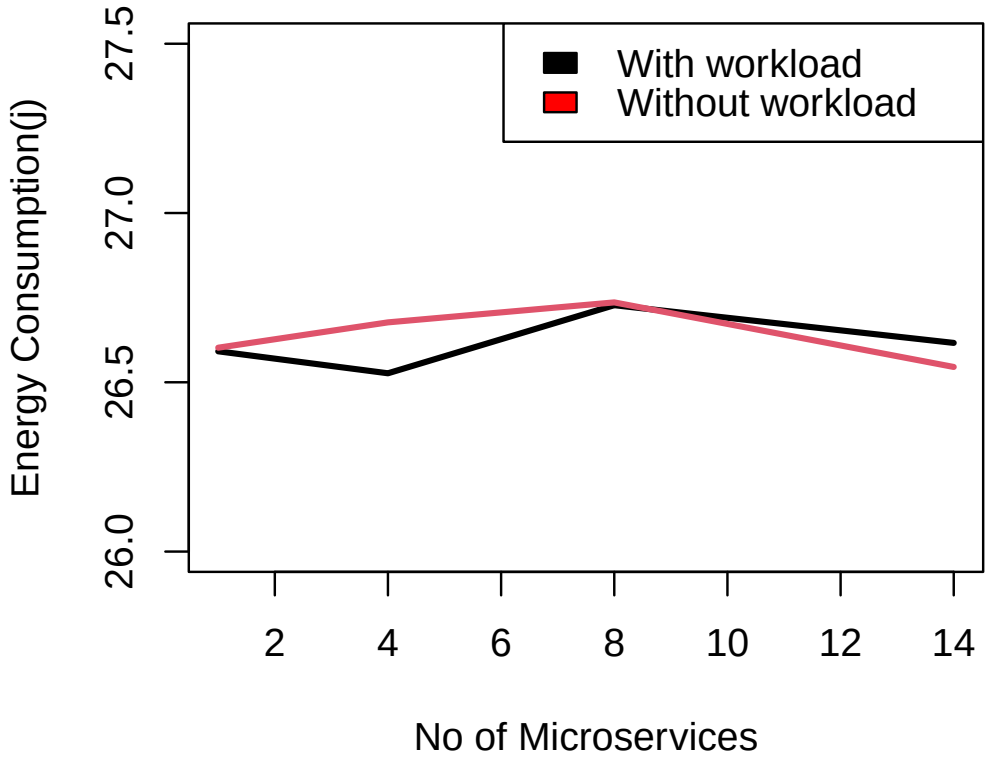


Figure 23: Line graph visualization of Number of microservices and energy usage by Carbon Aware KEDA Autoscalers with and without workload

of the microservice-based application (by **42.9%**) compared to the HPA-based autoscaler, resulting in an average energy reduction of **38.24%** per microservice.

We further analyzed the test results by creating a box plot for the energy consumption of each microservice in the sock shop microservice-based application. The energy consumption difference across the application was found to be consistent with the energy consumption comparison of each individual microservice (Figure 24). Microservices with extensive workloads showed a significant energy consumption difference, suggesting that the

Figure 24: Box Plot visualization of all microservices and their energy usage under both Carbon Aware KEDA and HPA Based Autoscalers

carbon-aware KEDA is well-suited for microservices with heavy workloads compared to those with lower workloads.

Finding 2

The Carbon Aware KEDA autoscaler demonstrates notably lower energy consumption for the microservice-based application compared to the HPA-based autoscaler, with a particular advantage observed for microservices experiencing heavy workloads in contrast to those with lighter workloads.

RQ3: How does the Carbon-Aware KEDA Autoscaling impact the response time of the microservices in the public cloud?

RQ3.1: Is there a statistically significant correlation between the energy consumption and response time of the microservice-based application when using either of the autoscalers?

To answer the RQs, Two separate hypotheses were tested in this study. The first hypothesis, H01, examined the impact on response time, while the second hypothesis, H02, investi-

gated the correlation between energy consumption and response time in the microservices-based application. The results of the Welch Two Sample t-test and Pearson's product-moment correlation coefficient showed the rejection of both H01 and H02 respectively, indicating that the utilization of Carbon Aware KEDA autoscaling led to an average increase in the application's response time by **30.68%**. Furthermore, the findings suggested a negative correlation between energy savings and response time in the application.

Figure 25: Box Plot visualization of response time for both test scenarios

The impact on response time was thoroughly analyzed for both test case scenarios, and the results consistently demonstrated a significant increase in response time when employing the Carbon Aware KEDA autoscaler in the microservices-based application (Figure 25). Due to the notable impact on response time, the Carbon Aware KEDA autoscaler is best suited for low priority workloads that do not require real-time processing, such as batch processing or machine learning model training.

Finding 3

The significant impact on response time makes the Carbon Aware KEDA autoscaler well-suited for handling low-priority workloads that do not necessitate real-time processing, such as batch processing or machine learning model training.

7.2 Sustainability Impact Assessment

The primary objective of this study is to contribute to the ongoing endeavor of achieving environmental sustainability in cloud computing. To achieve this goal, we have developed a carbon-aware autoscaling approach for the horizontal pod autoscaling policy in microservice-based applications deployed on the public cloud. We conducted empirical experiments to quantitatively analyze the approach's impact on energy consumption and response time of the microservice-based application. Additionally, in this section we have thoroughly examined the potential sustainability impact of the software artifact produced in this research.

To conduct a sustainability impact assessment of the carbon-aware autoscaler, we employed Sustainability Awareness Framework (SusAF), a framework designed to assist researchers and practitioners in evaluating the direct, enabling, and systemic effects of social-technical systems. The framework also takes into account various dimensions of sustainability, including individual, social, environmental, economic, and technical aspects.[65] By utilizing SusAF, we are able to comprehensively analyze the sustainability implications of the carbon-aware autoscaling approach.

Figure 26 illustrates two immediate impacts, one positive and one negative, on the environmental dimension of sustainability. As discussed in Section 7.1, the carbon-aware scaler components introduce their own energy overhead like any piece of software, which can be considered a negative aspect. However, in the broader context, these components contribute significantly to reducing the overall energy consumption of microservice-based applications, aligning with the primary objective of the autoscaling approach. This immediate impact facilitates carbon emission reduction (enabling impact), leading to a systemic impact in achieving environmental sustainability.

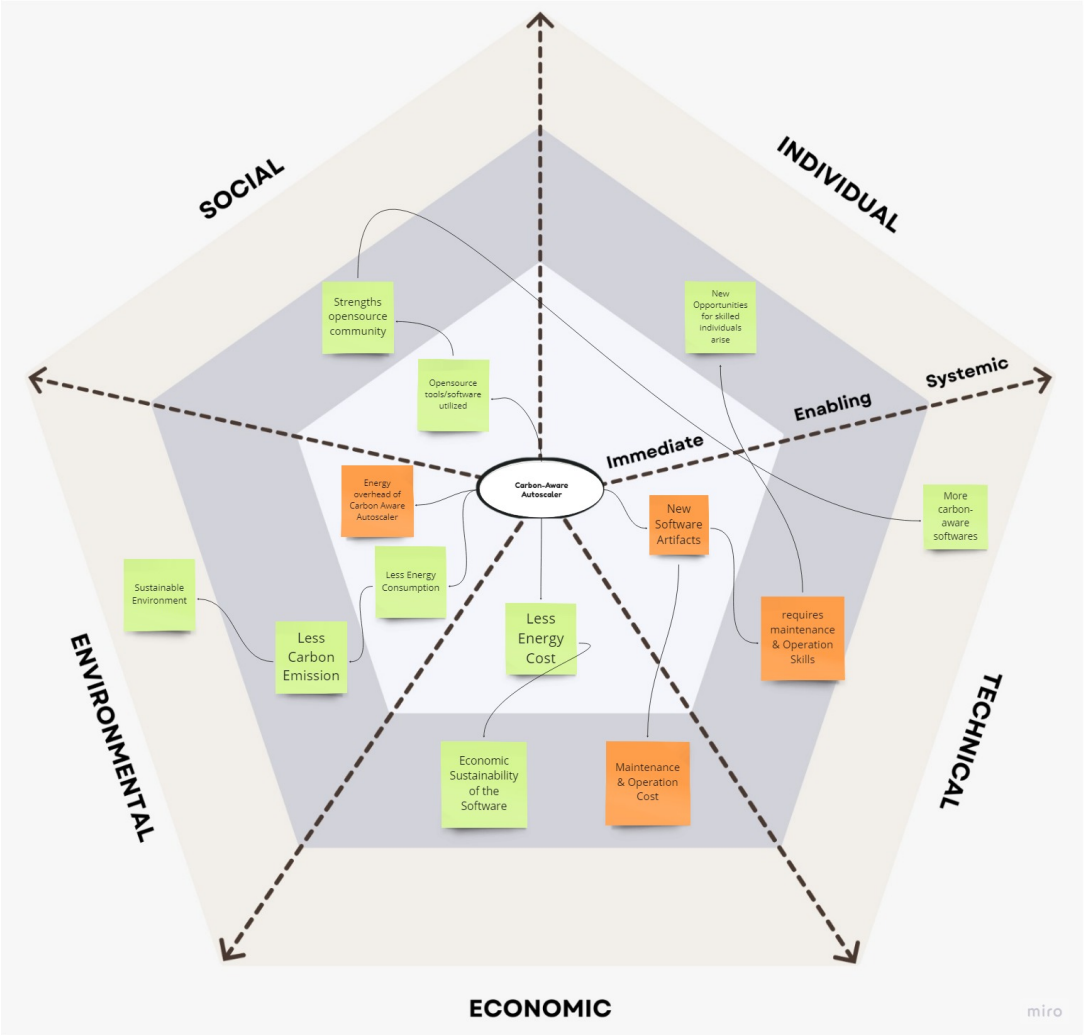


Figure 26: SusAF visualization of the Carbon Aware Autoscaler Sustainability Impact

Additionally, Figure 26 provides further insight into the immediate impact on the economic sustainability dimension. The carbon-aware autoscaling approach facilitates a reduction in energy costs, promoting the economic sustainability of software products leveraging this approach. On the technical sustainability dimension, the introduction of a new software artifact entails the need for maintenance and operational skills, which has associated economic costs. While this aspect may be perceived as a negative impact on the economic

dimension of sustainability, it can also be viewed positively as an enabling impact that creates new opportunities for skilled engineers, enhancing individual sustainability.

Moreover, the autoscaler is built using several open-source cloud-native technologies and is licensed as open-source on GitHub. This has an immediate impact on the social dimension of sustainability, fostering collaboration and contributing to the strength of the open-source community, thus having an enabling impact. Finally, it promotes the development of more carbon-aware software, leading to a systemic impact on the technical dimension of sustainability with increased adoption and development of sustainable software solutions.

8 Threats To Validity

In this section, we assess various potential threats to the validity of our experiment, using the classification framework proposed by Cook and Campbell[66]. This framework categorizes validity threats into four distinct components: internal, external, construct, and conclusion validity.

8.1 Internal Threats

Internal threats to validity in our study encompass maintaining a consistent and stable experimental environment, as well as addressing potential variability in the experimental results. To tackle these concerns, we utilized Terraform to ensure identical Kubernetes clusters for both autoscaling strategies, promoting a uniform and stable setup. Conducting multiple trials with diverse workload scenarios further enhanced the reliability and robustness of our analysis, mitigating the impact of inherent variability and reducing the risk of drawing biased conclusions. These measures bolstered the credibility of our findings, providing a comprehensive understanding of the impact of the autoscaling strategies on the microservice-based application's performance and energy consumption.

8.2 External Threats

An external threat to the validity of our study pertains to the generalizability of our findings to a broader range of real-world scenarios. To address this concern, we carefully selected a cloud infrastructure that ranks 2nd in terms of market share among public cloud providers. Additionally, we chose a representative microservice-based application exemplar for our experiment. However, it is important to note that the selected microservice-based application is not a real-world application. To mitigate this threat, we recommend replicating this experiment with real-world microservice-based applications that are commonly encountered in practical cloud environments. Conducting the study with diverse and authentic

applications can enhance the external validity of our findings, enabling more confident generalizations and inferences about the impact of the investigated autoscaling strategies on different real-world scenarios. By considering a broader range of applications and cloud infrastructures, future research can provide a more comprehensive understanding of the implications and practical implications of the studied autoscaling techniques.

8.3 Construct Threats

To ensure the construct validity of our study, we took measures to address potential threats. We provided a detailed explication of constructs in Chapter 3, outlining the factors and research questions, and logically selected treatments and subjects in Section **??**, ensuring a well-structured experimental design. To enhance the reliability and validity of our measurements, we utilized Kepler for granular energy consumption measurement and Locust for accurate response time measurement, thus bolstering the overall construct validity of our investigation.

8.4 Conclusion Threats

To mitigate the conclusion validity threat in our experiment, we employed rigorous statistical analysis and utilized appropriate evaluation methods. By conducting multiple trials of the experiment with different workload scenarios, we aimed to obtain consistent and reliable results. Additionally, we used appropriate statistical tests, such as the Welch Two Sample t-test and Cohen's d effect size test, to compare the performance of the autoscaling strategies accurately. Moreover, we ensured that the sample size was adequate and representative of the population under study, which further contributes to the conclusion validity of our findings. By addressing these aspects, we sought to minimize any potential threats to the validity of the conclusions drawn from our experiment on the impact of autoscaling strategies for microservice-based applications with Kubernetes in the public cloud.

9 Conclusion

This empirical experiment examines the impact of carbon aware autoscaling on microservices based applications in the public cloud. We utilized two autoscaling treatments: HPA-based and Carbon Aware KEDA, employing both write-intensive and read-intensive workload scenarios. The analysis of the experiment results revealed the following key findings: ***(i)***The Carbon Aware KEDA autoscaler exhibits higher energy cost efficiency, making it advantageous for managing larger quantities of microservices, ***(ii)***The Carbon Aware KEDA autoscaler demonstrates significantly lower energy consumption compared to the HPA-based autoscaler, especially for microservices under heavy workloads, ***(iii)***The Carbon Aware KEDA autoscaler is well-suited for low-priority workloads that do not require real-time processing, such as batch processing or machine learning model training, due to its positive impact on response time.

In future studies, extending the experiment to workload scheduling based on spatial and temporal carbon intensity values could provide valuable insights. Additionally, testing the approach with real-world microservices-based applications that do not necessitate real-time processing could further validate its effectiveness.

REFERENCES

[1] European Commission, *Corporate sustainability reporting*, URL: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32022L2464, Accessed: 16.12.2022.

[2] E. Gelenbe and Y. Caseau, "The impact of information technology on energy consumption and carbon emissions," *Ubiquity*, vol. 2015, no. June, Jun. 2015. DOI: 10.1145/2755977. [Online]. Available: https://doi.org/10.1145/2755977.

[3] I. Fé, R. Matos, J. Dantas, *et al.*, "Performance-cost trade-off in auto-scaling mechanisms for cloud computing," *Sensors (Basel)*, vol. 22, no. 3, p. 1221, Feb. 2022. DOI: 10.3390/s22031221.

[4] P. Singh, P. Gupta, K. Jyoti, and A. Nayyar, "Research on auto-scaling of web applications in cloud: Survey, trends and future directions," *Scalable Computing: Practice and Experience*, vol. 20, pp. 399–432, May 2019. DOI: 10.12694/scpe.v20i2.1537.

[5] L. M. N'dri, M. Islam, and M. Kakinaka, "Ict and environmental sustainability: Any differences in developing countries?" *Journal of Cleaner Production*, vol. 297, p. 126 642, 2021, ISSN: 0959-6526. DOI: https://doi.org/10.1016/j.jclepro.2021.126642. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0959652621008623.

[6] W. A. Hanafy, Q. Liang, N. Bashir, D. E. Irwin, and P. J. Shenoy, "Carbonscaler: Leveraging cloud workload elasticity for optimizing carbon-efficiency," *ArXiv*, vol. abs/2302.08681, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:257020069.

[7] A. James and D. Schien, "A low carbon kubernetes scheduler," in *ICT for Sustainability*, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:195693946.

[8] Green Software Foundation, *Software carbon intensity specification (sci)*, https://github.com/Green-Software-Foundation/sci/blob/dev/Software_Carbon_Intensity/Software_Carbon_Intensity_Specification.md, 2023.

[9] Microsoft Learn, *The principles of sustainable software engineering*, Microsoft Learn, 2023. [Online]. Available: https://learn.microsoft.com/en-us/training/modules/sustainable-software-engineering-overview/.

[10] Microsoft. "Core philosophies of sustainable software engineering." (Last accessed: May 2023), Microsoft Learn. (2021).

[11] K. Kinsiveer. "Sustainable software engineering." (Last accessed: May 2023), Helmes. (unknown).

[12] Microsoft. "Sustainable software engineering overview." (Last accessed: May 2023), Microsoft Learn. (unknown).

[13] G. Learn. "Principles of sustainable software engineering." (Last accessed: May 2023), Green Software Foundation. (unknown).

[14] Microsoft. "Carbon efficiency principle." (Last accessed: May 2023), Microsoft Learn. (unknown).

[15] G. Learn. "Carbon efficiency principle." (Last accessed: May 2023), Green Software Foundation. (unknown).

[16] Microsoft. "Energy efficiency principle." (Last accessed: May 2023), Microsoft Learn. (unknown).

[17] G. Learn. "Energy efficiency principle." (Last accessed: May 2023), Green Software Foundation. (unknown).

[18] Microsoft. "Carbon awareness principle." (Last accessed: May 2023), Microsoft Learn. (unknown).

[19] G. Learn. "Carbon awareness principle." (Last accessed: May 2023), Green Software Foundation. (unknown).

[20] Microsoft. "Hardware efficiency principle." (Last accessed: May 2023), Microsoft Learn. (unknown).

[21] G. Learn. "Hardware efficiency principle." (Last accessed: May 2023), Green Software Foundation. (unknown).

[22] G. Learn. “Measurement principle.” (Last accessed: May 2023), Green Software Foundation. (unknown).

[23] T. Anquetin, G. Coqueret, B. Tavin, and L. Welgryn, “Scopes of carbon emissions and their impact on green portfolios,” *Economic Modelling*, vol. 115, p. 105 951, 2022. DOI: 10.1016/j.econmod.2022.105951.

[24] G. Learn. “Climate commitment principle.” (Last accessed: May 2023), Green Software Foundation. (unknown).

[25] M. Learn, *Principle climate commitment*, https://learn.microsoft.com/en-us/training/modules/sustainable-software-engineering-overview/8-climate-commitments, Last accessed: May 2023.

[26] E. Radhika and G. Sudha Sadasivam, “A review on prediction based autoscaling techniques for heterogeneous applications in cloud environment,” *Materials Today: Proceedings*, vol. 45, pp. 2793–2800, 2021, International Conference on Advances in Materials Research - 2019, ISSN: 2214-7853. DOI: https://doi.org/10.1016/j.matpr.2020.11.789. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214785320394657.

[27] T. Chen and R. Bahsoon, “Self-adaptive and sensitivity-aware qos modeling for the cloud,” in *2013 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 2013, pp. 43–52. DOI: 10.1109/SEAMS.2013.6595491.

[28] Microsoft, *Autoscaling*, Microsoft Learn, 2023. [Online]. Available: https://learn.microsoft.com/en-us/training/modules/sustainable-software-engineering-overview/.

[29] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, “A performance analysis of ec2 cloud computing services for scientific computing,” in *Cloud Computing*, D. R. Avresky, M. Diaz, A. Bode, B. Ciciani, and E. Dekel, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 115–131, ISBN: 978-3-642-12636-9.

[30] T. T. Nguyen, Y. J. Yeom, T. Kim, D. H. Park, and S. Kim, “Horizontal pod autoscaling in kubernetes for elastic container orchestration,” *Sensors (Basel)*, vol. 20, no. 16, p. 4621, Aug. 2020. DOI: 10.3390/s20164621.

[31] M. Learn, *Autoscaling component - best practices for cloud architectures*, https://learn.microsoft.com/en-us/azure/architecture/best-practices/auto-scaling, Last accessed: May 2023, Microsoft.

[32] *Aws auto scaling types: Best practices*, https://www.developer.com/web-services/aws-auto-scaling-types-best-practices/, Last accessed: May 2023.

[33] V. Podolskiy, A. Jindal, and M. Gerndt, "Iaas reactive autoscaling performance challenges," Jul. 2018, pp. 954–957. DOI: 10.1109/CLOUD.2018.00144.

[34] N. Roy, A. Dubey, and A. Gokhale, "Efficient autoscaling in the cloud using predictive models for workload forecasting," in *2011 IEEE 4th International Conference on Cloud Computing*, 2011, pp. 500–507. DOI: 10.1109/CLOUD.2011.42.

[35] M. Abdullah, W. Iqbal, A. Erradi, and F. Bukhari, "Learning predictive autoscaling policies for cloud-hosted microservices using trace-driven modeling," in *2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2019, pp. 119–126. DOI: 10.1109/CloudCom.2019.00028.

[36] D.-H. LUONG, H.-T. THIEU, A. OUTTAGARTS, and Y. GHAMRI-DOUDANE, "Predictive autoscaling orchestration for cloud-native telecom microservices," in *2018 IEEE 5G World Forum (5GWF)*, 2018, pp. 153–158. DOI: 10.1109/5GWF.2018.8516950.

[37] M. Abdullah, W. Iqbal, A. Mahmood, F. Bukhari, and A. Erradi, "Predictive autoscaling of microservices hosted in fog microdata center," *IEEE Systems Journal*, vol. 15, no. 1, pp. 1275–1286, 2021. DOI: 10.1109/JSYST.2020.2997518.

[38] AWS. "Scheduled scaling for amazon ec2 auto scaling." Accessed: 28-August-2018, Amazon Web Services. (2018), [Online]. Available: https://docs.aws.amazon.com/autoscaling/ec2/userguide/scheduletime.html.

[39] S. El Kafhali, I. El Mir, K. Salah, *et al.*, "Dynamic scalability model for containerized cloud services," *Arabian Journal for Science and Engineering*, vol. 45, no. 12, pp. 10 693–10 708, 2020. DOI: 10.1007/s13369-020-04847-2. [Online]. Available: https://doi.org/10.1007/s13369-020-04847-2.

[40] J. Lewis and M. Fowler, *Microservices: A definition of this new architectural term*, https://www.martinfowler.com/articles/microservices.html, (Last accessed: May 2023), 2014.

[41] A. Cockroft, *Migrating to microservices*, https://youtu.be/1wiMLkXz26M, (Last accessed: May 2023), 2014.

[42] Y. Wang, H. Kadiyala, and J. Rubin, "Promises and challenges of microservices: An exploratory study," *Empirical Software Engineering*, vol. 26, no. 4, p. 63, May 2021, ISSN: 1573-7616. DOI: 10.1007/s10664-020-09910-y. [Online]. Available: https://doi.org/10.1007/s10664-020-09910-y.

[43] C. Richardson, *Microservice architecture*, https://microservices.io/, (Last accessed: July 2020), 2014.

[44] K. Beck, M. Beedle, A. van Bennekum, *et al.*, *Manifesto for agile software development*, https://agilemanifesto.org, (Last accessed: May 2023), 2001.

[45] L. Chen, "Microservices: Architecting for continuous delivery and devops," in *2018 IEEE International Conference on Software Architecture (ICSA)*, 2018, pp. 39–46.

[46] W. Luz, E. Agilar, M. C. de Oliveira, C. E. de Melo, G. Pinto, and R. Bonifácio, "An experience report on the adoption of microservices in three brazilian government institutions," in *Proceedings of Brazilian Symposium on Software Engineering (SBES)*, 2018, pp. 32–41.

[47] J. Soldani, D. A. Tamburri, and W.-J. van den Heuvel, "The pains and gains of microservices: A systematic grey literature review," *Journal of Systems and Software*, vol. 146, pp. 215–232, 2018.

[48] M. Viggiato, R. Terra, H. Rocha, M. T. Valente, and E. Figueiredo, "Microservices in practice: A survey study," in *Brazilian Workshop on Software Visualization, Evolution and Maintenance (VEM)*, 2018, pp. 1–8.

[49] A. Bucchiarone, N. Dragoni, S. Dustdar, S. T. Larsen, and M. Mazzara, "From monolithic to microservices: An experience report from the banking domain," *IEEE Software*, vol. 35, no. 3, pp. 50–55, 2018.

[50] J. Fritzsch, J. Bogner, S. Wagner, and A. Zimmermann, "Microservices migration in industry: Intentions, strategies, and challenges," in *Proceedings of the 2019 IEEE*

International Conference on Software Maintenance and Evolution (ICSME), 2019, pp. 481–490.

[51] D. Taibi, V. Lenarduzzi, and C. Pahl, "Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation," *IEEE Cloud Computing*, vol. 4, no. 5, pp. 22–32, 2017.

[52] I. Nardin, R. Righi, T. Lopes, C. André da Costa, H. Yeom, and H. Köstler, "On revisiting energy and performance in microservices applications: A cloud elasticity-driven approach," *Parallel Computing*, vol. 108, p. 102 858, Oct. 2021. DOI: 10.1016/j.parco.2021.102858.

[53] *Cloud autoscaling*, https://www.techtarget.com/searchcloudcomputing/definition/autoscaling, Last accessed May 2023.

[54] Microsoft Learn. "Carbon awareness." (Year of Access (e.g., 2023)).

[55] V. R. Basili, G. Caldiera, and D. Rombach, "The Goal Question Metric Approach," in *Encyclopedia of Software Engineering*, Wiley, 1994, pp. 528–532.

[56] S. S. S. N. Usha and K. Srinivasan, "A comparative study on three selective cloud providers," *International Journal on Cybernetics Informatics (IJCI)*, vol. 11, no. 4, pp. 167–178, 2022. DOI: 10.5121/ijci.2022.110413. [Online]. Available: https://www.ijcionline.com/paper/11/11422ijci13.pdf.

[57] *Carbon Aware KEDA Operator*, https://github.com/Azure/carbon-aware-keda-operator, Last accessed: May 2023.

[58] L. Ardito, R. Coppola, M. Morisio, M. Torchiano, and M. Risi, "Methodological guidelines for measuring energy consumption of software applications," *Sci. Program.*, vol. 2019, Jan. 2019, ISSN: 1058-9244. DOI: 10.1155/2019/5284645. [Online]. Available: https://doi.org/10.1155/2019/5284645.

[59] P. Mishra, C. M. Pandey, U. Singh, A. Gupta, C. Sahu, and A. Keshri, "Descriptive statistics and normality tests for statistical data," *Annals of Cardiac Anaesthesia*, vol. 22, no. 1, pp. 67–72, Jan. 2019. DOI: 10.4103/aca.ACA_157_18.

[60] Z. Hanusz, J. Tarasinska, and W. Zieliński, "Shapiro–wilk test with known mean," *Revstat Statistical Journal*, vol. 14, pp. 89–100, Feb. 2016. DOI: 10.57805/revstat.v14i1.180.

[61] E. Ostertagova and O. Ostertag, "Methodology and application of one-way anova," *American Journal of Mechanical Engineering*, vol. 1, pp. 256–261, Nov. 2013. DOI: 10.12691/ajme-1-7-21.

[62] Z. Lu and K.-H. Yuan, "Welch's t test," in Jan. 2010, pp. 1620–1623. DOI: 10.13140/RG.2.1.3057.9607.

[63] C. R. Brydges, "Effect Size Guidelines, Sample Size Calculations, and Statistical Power in Gerontology," *Innovation in Aging*, vol. 3, no. 4, igz036, Sep. 2019, ISSN: 2399-5300. DOI: 10.1093/geroni/igz036. eprint: https://academic.oup.com/innovateage/article-pdf/3/4/igz036/33010974/igz036.pdf. [Online]. Available: https://doi.org/10.1093/geroni/igz036.

[64] M.-T. Puth, M. Neuhäuser, and G. D. Ruxton, "Effective use of pearson's product–moment correlation coefficient," *Animal Behaviour*, vol. 93, pp. 183–189, 2014, ISSN: 0003-3472. DOI: 10.1016/j.anbehav.2014.05.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0003347214002127.

[65] B. Penzenstadler, N. Seyff, S. Betz, *et al.*, "Vision paper: The sustainability awareness framework (susaf) as a de-facto standard?" English, *CEUR Workshop Proceedings*, vol. 3378, Apr. 2023, Funding Information: Part of this work has been funded by the Area of Advance at Chalmers under project number 37460087, and the Dept. of Research and Universities (Catalonia Gov.) under Grant Ref. 2021 SGR 01396. Publisher Copyright: © 2023 Copyright © 2023 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0); Joint of REFSQ-2023 Workshops, Doctoral Symposium, Posters and Tools Track and Journal Early Feedback, REFSQ-JP 2023 ; Conference date: 17-04-2023 Through 20-04-2023, ISSN: 1613-0073.

[66] E. T. Koh and W. L. Owen, "Experimental and quasi-experimental research," in *Introduction to Nutrition and Health Research*. Boston, MA: Springer US, 2000, pp. 196–217, ISBN: 978-1-4615-1401-5. DOI: 10.1007/978-1-4615-1401-5_11. [Online]. Available: https://doi.org/10.1007/978-1-4615-1401-5_11.