National College of Ireland

Orchestrating Contextual Bandits Algorithm for Resource Scheduling in Kubernetes on Multiple Cloud Environments using Linear regression model

MSc Research Project
MSc Cloud Computing

Bramha Theja Gadikota
Student ID: X23197994

School of Computing
National College of Ireland

Supervisor: Shaguna Gupta

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Bramha Theja Gadikota

Student ID: X23197994……

Programme: MSc Cloud Computing **Year:** 2025
……

Module: MSc Research Project

Supervisor: Shaguna Gupta ……

Submission Due Date: 12/12/2024……

Project Title:

Orchestrating Contextual Bandits Algorithm for Resource Scheduling in Kubernetes on Multiple Cloud Environments using Linear regression model

Word Count: ……**Page Count:** ……..

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ……

Date: ……

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Orchestrating Contextual Bandits Algorithm for Resource Scheduling in Kubernetes on Multiple Cloud Environments using Linear regression model

Name: Bramha Theja Gadikota
X23197994@student.ncirl.ie
National College of Ireland

Abstract

Resource allocation in Kubernetes clusters refers to the efficient distribution and management of computing resources, such as CPU, memory, and storage to workloads running in the cluster. Traditional resource scheduling approaches often struggle with issues like suboptimal resource utilization, inability to handle dynamic workloads, and lack of adaptability to varying application demands. These problems are addressed in this study by employing the Contextual Bandits algorithm, which enables more intelligent and adaptive decision-making based on real-time resource usage patterns. The proposed approach uses a Linear Regression model within the Contextual Bandits framework to predict the most efficient allocation of resources based on historical data and the current context. The algorithm attempts to find a balance between maximizing resource efficiency and maintaining application performance. The objective of this study is to compare the performance of the proposed Linear Regression-based scheduling approach between AWS and Azure cloud platforms using Kubernetes, which uses a more static and simplistic allocation mechanism. The experimental results demonstrated that the proposed approach outperforms the scheduler in terms of resource utilization, application responsiveness, and scalability. The findings indicate that Linear Regression scheduling with Kubernetes can significantly enhance cloud resource management, offering improved performance in handling dynamic and fluctuating workloads compared to traditional methods.

Keywords: Cloud Computing, AWS (Amazon Web Services), Microsoft Azure, Kubernetes, Docker Containerization, Cloud

1. Introduction

1.1 Introduction

Resource scheduling in cloud computing means allocating and controlling computing resources including CPU, memory, and storage for performing several tasks or for running applications (Praveenchandar and Tamilarasi, 2021). It also confirms that the resources have been procured in the right quantity and quality that can fulfil the performance demands at the least cost possible and resource wastage. Kubernetes, which is an open-source container orchestration system, is one of the most popular platforms for managing and deploying containerized applications (Miller et al., 2021) for creating automated environments for them by offering several tools for deployment, scaling, and management of workloads across sets of machines. It allows dynamic resource scheduling and helps in the allocation of containers to the nodes depending on the number of resources required and certain policies. Kubernetes takes applications to another level of flexibility, scalability and failure support for scaling up

or down, load balancing and cluster support. On the other hand, cloud computing is a model that provides access to shared computation and storage based on the current need and provides services remotely over the internet. It enables organizations to gain computing resources, storage, and other services as needed without purchasing the associated hardware. When coupled with cloud technologies such as kubernetes, flexible planning of resources simplifies application deployment, scaling, and management for optimum use of resources and maximum availability. This integration helps businesses to address customer variability in their demand and needs while operating at their best efficiency without frequent stoppages.

1.2 Aim of the study

The aim of this study is to solve the resource allocation challenges in kubernetes clusters by using a linear regression model within the Contextual Bandits framework. Various problems result from the traditional approaches to resource scheduling including poor resource utilization, inflexibility, and their inability to handle dynamic loads and changing application workloads. These challenges are sought to be addressed in this study by planning for resource allocation proactively, that is, decision-making will not just be done based on past information but also through running experiments on possibilities of efficient resource use. To this end, the recommended approach uses the Linear Regression model to predict the optimal number of resources most fitting to the current system environment and its performance history to make a smarter and more sophisticated schedule decision. These classifications make it possible for the system to proactively adjust resources on the fly, increasing the efficiency of both resources and applications.

1.3 Research Questions

How can the integration of Contextual Bandits algorithms with Kubernetes resource scheduling on AWS and Azure optimize cloud infrastructure performance, and when should the algorithm dynamically adjust to achieve the most efficient CPU utilization between these multiple-cloud environments?

1.4 Research Objectives

The research questions for this report are:

1. How does the performance of the Linear Regression-based resource scheduling approach in terms of resource utilization and efficiency on AWS and Azure platforms?
2. What impact does adopting a proactive, data-driven resource allocation strategy, based on historical performance data and ongoing exploration, have on the scalability and responsiveness of applications in Kubernetes clusters on AWS and Azure?

1.5 Problem Statement

Kubernetes clusters struggle to handle efficient resource allocation and scheduling mainly because of changing workload patterns and multiple cloud environments. The use of conventional scheduling approaches results in poor performance outcomes that produce delayed responses and higher failure incidents and underused resources. Network complexity becomes more substantial in environments which use AWS alongside Azure because both clouds have separate underlying infrastructures which affect application protocols. An

analysis of these resource scheduling challenges employs contextual bandits algorithm combined with linear regression modeling to enhance resource scheduling operations. The evaluation of this algorithm across AWS and Azure cloud platforms helps researchers determine which platform offers optimal conditions for producing scalable and reliable Kubernetes deployments.

1.6 Motivation

Significant progress in using containers alongside multi-cloud systems requires reliable orchestration systems to achieve enhanced operational competence and dependable system performance. Although highly efficient Kubernetes encounters limitations while managing resources across changing workload requirements. This research incorporates contextual bandits algorithms to boost scheduling efficiency because traditional frameworks required improvement. This comparative investigation of AWS and Azure offers tangible findings about major cloud platforms which help organizations decide their optimal infrastructure selection. Research findings affirm their crucial role in advancing cloud deployments due to their ability for performance improvement and scaling benefits.

1.7 Structure of the Report

This section is going to explain the structure of the report which is as follows:

Chapter 1 Introduction: Provides an overview of the study, its objectives, and the significance of cloud-based resource optimization and load testing.

Chapter 2 Literature Review (LR): Summarizes previous research on cloud platforms, Kubernetes, and resource management strategies to establish the study's context.

Chapter 3 Methodology: Describes the approach taken, including tools and frameworks like Locust, Kubernetes, Prometheus, and Grafana for experimentation.

Chapter 4 Design Specification: Outlines the system architecture, key components, and configurations for resource management and monitoring.

Chapter 5 Implementation: Details the practical deployment of AWS and Azure platforms, Kubernetes clusters, and monitoring setups.

Chapter 6 Evaluation: Analyzes performance metrics like CPU, memory, and network usage under load, comparing AWS and Azure scalability and reliability.

Chapter 7 Conclusion and Future Works: Summarizes findings, highlighting the strengths and weaknesses of both platforms, and discusses the study's contributions and also showing future works.

2. Literature Review

2.1 Contextual Bandits Algorithm

The contextual bandit's algorithm is more advanced version of reinforcement learning which was developed based on the multi-armed bandit problem (Bouneffouf et al., 2020), but the context information is considered during the decision-making process. While conventional multi-armed bandits are a two-stage technique of maximizing reward that combines exploration and exploitation throughout a series of trials, contextual bandits work to enhance the capacity to choose the right action from the context of the decision situation. This context-aware framework allows algorithm optimization for various situations (Islam et al., 2021), which is a major reason why it is optimal for practical uses such as product recommendations and dynamic advertising techniques as well as optimization of resource utilization in organizations. In each time step, the algorithm measures a certain set of contextual features or equivalencies, chooses an action out of a predetermined set and receives a score based on the utility of the action selected. In the long run, it acquires a policy that takes a context and an action and yields the most reward. It is less sensitive to the choice of reward functions and an important feature of contextual bandits that it can work in the environment in which reward-to-some context follows a non-stationary distribution. The learning process usually includes estimating all the action-value functions for every context; some of the plans are linear regression or neural networks. Council nonetheless contextual bandits have issues for example; the efficient exploration policies that should be employed to prevent bad decisions and computational tractability at circumstances with many features (Bietti et al., 2021).

There are some recent advanced which have seen algorithms such as LinUCB, or Linear Upper Confidence Bound that helps in easing the exploration-exploitation conundrum and increases efficiency. In the field of decision making about resource allocation, contextual bandits are a promising solution for making resource schedules depending on workloads and system status. They facilitate control decisions in data environments within distributed systems such as Kubernetes, where context includes systems like CPU consumption, memory requirements, and latency. When using contextual bandits, resource scheduling processes are made intelligent, leading to less resource wastage and increased resource utilization. Additionally, it shows that learning from other models such as linear regression can improve contextual bandits' understanding of those context feature, which in turn will lead to more better and specific decisions. That is why contextual bandits have a very high application potential in multi-cloud environments where resources and network conditions may vary significantly, and therefore require reliable and smart scheduling. In general, contextual bandits can be considered as a promising and highly universal formalism that unifies theory and practice of dynamic decision making with the usage of contextual information.

2.2 Kubernetes and Resource Scheduling

2.2.1 Kubernetes Overview

Kubernetes, a container orchestration software, is now an industry standard for managing applications based on containers across a variety of contexts (Casalicchio and Iannucci, 2020). Initially built by Google and later open-sourced and handed over to the Cloud Native Computing Foundation (CNCF) (Vano et al., 2023), Kubernetes is an orchestration tool meant to manage containers. It offers a strong foundation for the administration of clusters of virtual machines and also meeting server app demands. Fundamentally, Kubernetes

introduces a declarative model by which users can define the applications, allowing the system to consistently maintain the given state. Components of Kubernetes are the master node involved in control plane tasks like scheduling or resource allocation and the worker nodes: containers within pods— the most basic executable structures in Kubernetes. The kube scheduler is another element of the control plane and is responsible for deciding which node should be utilized to run new pods (Qunaibi, 2023) in regard of aspects such as resource availability, policies and affinity. Another advantage of Kubernetes is its flexibility which lets developers create their scheduling algorithms and integrate third-party programmes for certain applications. Besides this, the ability to perform service discovery, load balancing and self-healing also makes it a good system to manage complex distributed systems. Concerning cloud computing as the type of computing that has recently emerged.

Pave the way to multi-cloud and hybrid-cloud arrangement of the application, Kubernetes has a strength where the underlying infrastructure is hidden from other applications, hence, prepare the way for portability by avoiding product lock-in (Schumann, 2024). This capability is especially useful in high volume situations, where traffic needs to be divided across several providers of cloud computing services. The possibility of scaling applications both up and out guarantees the optimal use of the resources thus keeping the operating costs low. Though, the starting scheduling algorithms are most efficient and fair, and do the best optimizing the resource usage they do not always cope with real-time and/or nonlinear requirements that can appear in multi-cloud environment. Such limitations point to the necessity of incorporating contextual bandits, as higher-level machine learning procedures in the scheduling of resources. Such innovations can be easily implemented on the platform due to the built-in APIs and plugins support, making it the best base for modern efficient resource management approaches. A large number of applications and services along with a support community makes Kubernetes dynamic and able to adapt to the constantly changing environment of cloud computing. Due to the ability to hide infrastructure details and provide a straightforward way to manage container packaging and distribution, Kubernetes helps now organizations to focus on respective application and services, unleashing process and thus innovation potential.

2.2.2 Existing Scheduling Algorithms

There is first study which is given by (Carrion, 2022) who has done extensive work on analyzing the Kubernetes scheduling techniques, especially considering the physical resources assignment to containers normally know as Scheduling Policies of Quality of Service (QoS) such as response time, energy efficiency, and resource consumption. The work introduces a novel classification schema for Kubernetes scheduling and systematically reviews the prior empirical findings in this area to find the shortcoming and prospect for future advancement. To this end, the proposed approach describes the current scheduling methods in a systematic way and inform future advancements on Kubernetes orchestration. However, the study also encountered difficulties in achieving the scheduling strategies agglomeration because of the multiple approaches combined with the high variance of QoS parameters as well as the working in the containerized environments. The outcomes are useful in supporting the notion that increasingly sophisticated and pragmatic scheduling schedules are necessary for addressing compensation and resource utilization overheads. Although the study is effective in interpreting what exists and what the consequences of these methodologies are, the reliance in empirical data restricts the degree of empirically based testing and application of proposed solutions. To overcome these limitations of the current study, the future research could include the real-life testing of the proposed solution and

could also investigate the possibility of the synergy between Kubernetes scheduling and other strategies.

(Menouer, 2021) has developed a Kubernetes Container Scheduling Strategy (KCSS) which will enable the scheduler to meet the user goal of minimizing makespan while at the same time ensuring the cloud provider goal of low power consumption is achieved. Unlike existing single-criterion scheduling strategies, KCSS adopts a multi-criteria approach using six key metrics: It includes CPU usage, memory usage, disk usage, total power consumption of the hardware, number of total running containers, time taken for image transmission. Based on the fundamental concept of the TOPSIS algorithm, KCSS consolidates these criteria to identify an optimal node for a newly arrived container. Since KCSS is implemented in the Go language and required few modifications to the Kubernetes framework, it exhibits better performance than traditional approaches in multiple situations. However, the approach has its shortcomings as far as scalability of multiple-criteria decision-making computations and flexibility to work in a changing cloud environment with fluctuating workloads. Despite the contribution of the study and the improvement in scheduling performance, the limitation with the approach lies in the fact that it is based on predefined criteria and the fact that the integration of KCSS with other Kubernetes updates may be cumbersome especially in complex cloud infrastructures or more diverse infrastructures. With regards to this study's future work, the selection criteria could be refined further and the workload and infrastructure conditions under which the strategy is tested can be expanded.

Based on Kubernetes, (Lovenvald, 2021) presented a serverless distributed deep learning job handler using Kubernetes to compare the default scheduler, and a gang-like custom scheduler in terms of job time. The purpose of this study was, therefore, to assess the suitability of these schedulers in managing the distribution of computing resources for training the deep learning models. From the experimented set up involving distinct deep learning models, resource numbers, and parallel jobs, it is evident that job run times were not altered exponentially. However, the gang scheduler demonstrated two notable benefits: In particular, it reduces the so-called resource deadlock, where resources are claimed but no jobs can begin; it also lessens epoch stragglers, where one or more jobs with a limited number of workers delays the completion of epochs and blocks other jobs' use of the resources. Thus, despite the revealed benefits, the study encountered difficulties in comparing performance based on the limited number of given scenarios, and it might not be applicable for greater and more composite working loads. The outcomes indicate that the proposed gang scheduler increases resource utilisation and that further studies should be carried out to evaluate the effectiveness of the gang scheduling in other cloud platforms and tasks.

(Rejiba and Chamanara, 2022) also used a survey to systematically review and categorize the existing works on custom Kubernetes Schedulers towards highlight the new age application areas such as AI-ML-DL and edge computing tasks. It has sought to present a mapping of scheduling objectives, workload kinds, and environmental settings targeted in the literature, so that prospective researchers may identify key scholarship in different areas and organize a diverse variety of scheduling goals and objectives. That said, the survey aimed at filling existing knowledge gaps to make informed recommendations for further research and development in Kubernetes scheduling. The results presented various approaches developed specifically to cover weaknesses in the default Kubernetes scheduler, which cannot perform well in handling complex workloads. The first difficulty was the integration of a wide-ranging categorisation due to different nature of scheduling needs and approaches in the studies. A major weakness of the survey approach is that it lacks the ability to unveil new and unpublished trends and opportunities in the field. Furthermore, because Kubernetes' services

and their ecosystems are still growing it can be challenging to keep survey and its results up to date.

Finally (Forre, 2022), has been put forward with the Resource-Optimized-Software-Testing (ROST) algorithm resulting in optimization of resource usage in continuous integration systems through bin packing strategy for scheduling of jobs. This work considered worst-fit, best-fit, next-fit and first-fit bin-packing algorithms and combined them with the ROST method that selects the packing strategy depending on bin capacity. This approach integrates the principle of scheduling the best job into the lot together with worst-fit and best-fit algorithms in an attempt to overcome complexities that characterize testing phases of software development. The experiments indicated that ROST produces less interrupts than traditional methods and is substantially more effective than the most prevalent approaches to CI scheduling. However, the algorithm design and its deployment were hampered by a trade-off between complexity and efficiency, especially with respect to dynamic load adaptability. As depicted in the results, ROST outperformed other schemes in tested scenarios but the observed results are conditioned by certain workload patterns and ought to be further checked in different software testing environments and large-scale systems.

This study presented by (Hanna et al., 2023) on the stochastic contextual linear bandit problem in which the objective is to develop algorithms capable of playing nearly optimally in spite of changing contexts where actions are varying and their associated rewards are random variables drawn according to the inner product of the action and an unknown parameter. The contextual problem introduces more difficulty than the classic and singular linear bandit problem due to stochastic nature of the contexts. To solve these problems the authors present a new reduction approach that relies on the given context distribution in order to reduce the stochastic contextual linear bandit problems into instances of linear bandit problems. Otherwise for the unknown context distributions they propose an algorithm that transform the learning problem into a sequence of first-order linear bandit instances with minimal error in the projections and by doing so, they move to the next stratum in addressing open challenges with high probability regret bound. Their approach shows promising enhancements of remorse restrain for distinct applications; in batch arrangements, completely misspecified contexts, spare parameters, and adversarial contamination. Nevertheless, there are certain drawbacks of the approach: The first one is that the context distribution is supposed to be known or approximate, which might not be true in many real-life cases. However, the proposed framework serves as a nice framework for overcoming stochastic contextual bandit issues and can be useful for inferring how regret bounds can be optimized under such settings.

Table 1: Comparison Table

Author (Year)	Key Features	Strengths	Weaknesses	Proposed Algorithm/Approach
(Carrion, 2022)	Focus on Kubernetes scheduling for containerized workloads. Introduces a new taxonomy for Kubernetes scheduling.	Provides a structured understanding of scheduling techniques, highlights gaps and opportunities for future research.	Limited experimental validation and real-world testing.	New taxonomy for Kubernetes scheduling, empirical study of existing techniques.

(Menouer, 2021)	Introduces a multi-criteria container scheduling strategy (KCSS) for Kubernetes using TOPSIS.	Improves scheduling performance, reduces power consumption and optimizes resource allocation.	Computational overhead of multi-criteria decision-making; challenges in integration with diverse workloads.	Kubernetes Container Scheduling Strategy (KCSS) using TOPSIS algorithm.
(Lovenvald, 2021)	Comparison of default Kubernetes scheduler with a gang-like custom scheduler for distributed deep learning jobs.	Identified benefits of the gang scheduler: prevents resource deadlocks and reduces epoch straggling.	No significant difference in job completion times; limited test scenarios.	Serverless distributed deep learning job handler using Kubernetes and custom gang scheduler.
(Rejiba and Chamanara, 2022)	Survey on custom Kubernetes schedulers for emerging applications (e.g., machine learning, edge computing). Provides a taxonomy of scheduling objectives and workload types.	Comprehensive classification of scheduling methods, highlights research gaps.	Limited by existing literature; rapidly evolving Kubernetes ecosystem may make findings outdated.	Survey of custom Kubernetes schedulers and taxonomy of approaches.
(Forre, 2022)	Development of ROST (Resource-Optimized-Software-Testing) algorithm combining bin-packing and optimal job scheduling.	Hybrid approach improves resource utilization in continuous integration systems; performs better than common approaches.	Complexity in balancing efficiency with dynamic workloads; limited generalizability across environments.	Resource-Optimized-Software-Testing (ROST) algorithm for software testing in continuous integration.
(Hanna et al., 2023)	Stochastic contextual linear bandit problem.	Reduction of stochastic contextual bandits to linear	Assumes known or easily estimable context	Reduction framework converting stochastic contextual bandit problems to linear

	Context-dependent actions with rewards as inner product of actions and unknown parameters-Novel reduction framework	bandits.	distributions may face challenges in real-world scenarios with complex, unknown distributions	bandit instances algorithm for handling unknown context distributions with minimal misspecifications

3. Research Methodology

This research methodology for proactive resource scheduling in Kubernetes is going to focus on optimizing resource allocation for achieving good use of CPU and memory which will minimize underutilization as well as overutilization. The first step in this methodology is based on the assessment of load metrics of various applications, launched in Kubernetes. This includes the process of determining a number of resources needed at per container or pod by checking on factors such as CPU usage, memory usage or the amount of input/output that the containers require. Based on this data, we deploy a forecasting model anchored on linear regression that estimates the resource consumption profile of each workload in the long run. These have historical CPU utilization data used to train the model to predict future resource requirements in the CPU. To achieve greater scalability and flexibility the model is wrapped into docker containers so it could be easily deployed on different environments. The deployment of this environment uses Amazon Web Service (AWS) with Elastic Container Registry (ECR) for pushing docker images for centralized image management and Elastic Kubernetes Service (EKS) for deploying the application to an EKS cluster. Whereas for Azure deployment, Azure Container Registry (ACR) is used to push the docker image and Azure Kubernetes Service (AKS) for deploying application to EKS cluster. This is done with the help of the outputs of the predictive model, which identifies the necessary updates in the resource and prevents them from becoming underutilized and overused when they are not needed. To improve on the usage of resources within the applications, Kubernetes horizontal pod autoscaling and resource quotas are used for the tuning of distribution. For the monitoring and visualization of the resource usage of containers, Prometheus and Grafana take care of collecting resources such as CPU in real-time informing the performance of the system. This makes it possible to effectively manage and allocate resources, minimize resource utilization and to guarantee that workloads in dynamic cloud-native settings will run optimally.

3.1 Contextual Bandits Algorithm for Resource Scheduling

The Contextual Bandits Algorithm has turned out to be reliable in the challenging circumstances of resource scheduling in Kubernetes environments that function across multiple clouds while prioritizing efficiency, scalability, and improved cost control. Due to its linear regression model, there is a clear reduction of the number of policy options for a given set of contextual states thus facilitating a quick assessment of the likely course of action. The inclusion of this approach in Kubernetes, a container orchestration platform increases the system's ability to efficiently balance tasks across dissimilar cloud structures, reduce latency, and optimize resource utilization. As opposed to systems of fixed scheduling

or rule-based systems, the Contextual Bandits Algorithm learns from feedback hence suitable for dynamic and unpredictable cloud workloads. Its capability to explore (search for better scheduling techniques) and to exploit (implement known good scheduling strategies) brings a nearly optimal distribution of computational resources. This flexibility translates to lower operations expenses when picking cost-effective nodes in multi-cloud settings, notwithstanding high performance. Furthermore, scalable is another requirement of the current cloud-native applications that the algorithm satisfies since it provides resources to the pods depending on the priority, resource request, and throughput. This is even the case in Kubernetes where applications run across multiple cloud providers and each provider charges differently and offers different performance.

The linear regression model serves as a computational corpus for determining the consequences of scheduling decisions which makes it efficient in replacing the overhead usually implicated by more diversified machine learning algorithms. This methodological choice allows keeping the algorithm itself minimally invasive to Kubernetes, thus preserving the complexity factor inherent to the platform while not hindering its efficiency. The orchestration of the Contextual Bandits Algorithm, it presents the idea of achieving a higher level of efficiency at lower cost, reduced latency and scalability of resources to help organizations attain significantly higher levels of performance to get the most out of their cloud applications. The approach makes resource allocation to solve multi-cloud strategic objectives, making it a revolutionary solution in cloud computing.

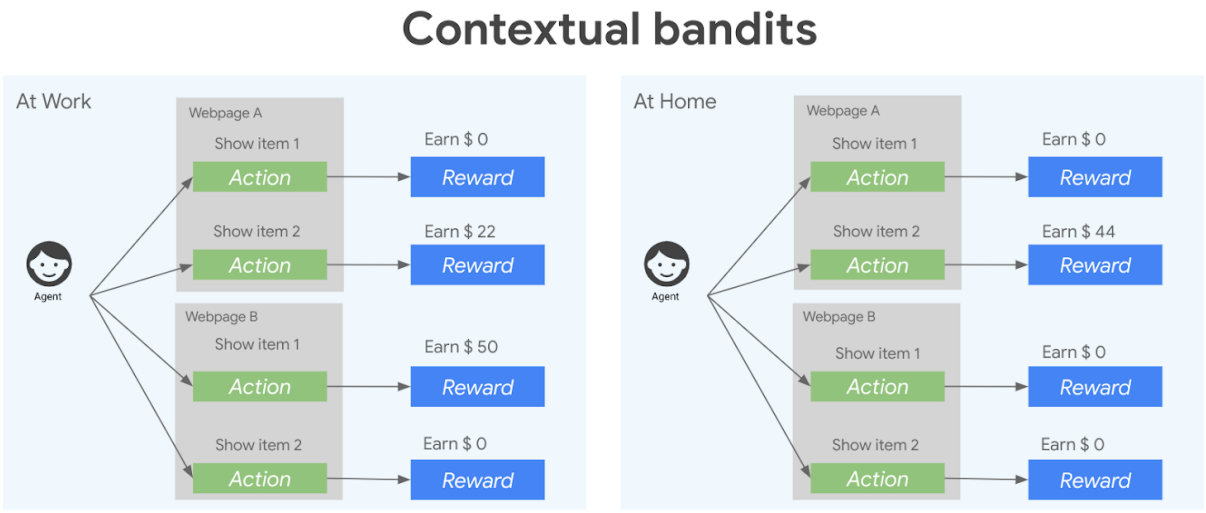

Figure 1: Contextual Bandits Algorithm by (https://cloud.google.com/blog/products/ai-machine-learning/how-to-build-better-contextual-bandits-machine-learning-models**)**

3.2 Research Methodology Flowchart

This project aims at deploying a fully functional and cost optimize solution using Amazon AWS services including EKS and an EC2. To accomplish the system's predictive modelling technique and for the decision making, flask framework and numpy python library were used to implement linear regression algorithm. everything was containerized by using docker, the docker image was being created and deployed into a containerized environment through docker commands within the EKS cluster so that it remains portable. Originally, a custom controller was incorporated into working application from which application performance could be controlled and monitored easily. To monitor the resource usage and system performance, Prometheus, open-source monitoring solution was used to gather the real time

metrics like memory and CPU usage from deployed instances. The collected metric was then analyzed using grafana the leading visualization tool which gave an insight of the metric through custom queries in graph format. Such visualization allowed us to look at important measurements in detail and to compare them before and after some changes were made to make sure that the systems are being used efficiently and that they would maintain their performance whether they were under load or not. The solution adopted AWS cloud infrastructure, containerization of the algorithm and data, and real-time monitoring tools and visualizations leading to a well-constructed, solution with scalability to support the deployment and performance monitoring of the linear regression algorithm that leveraged the resource effective nature of the cloud native core.

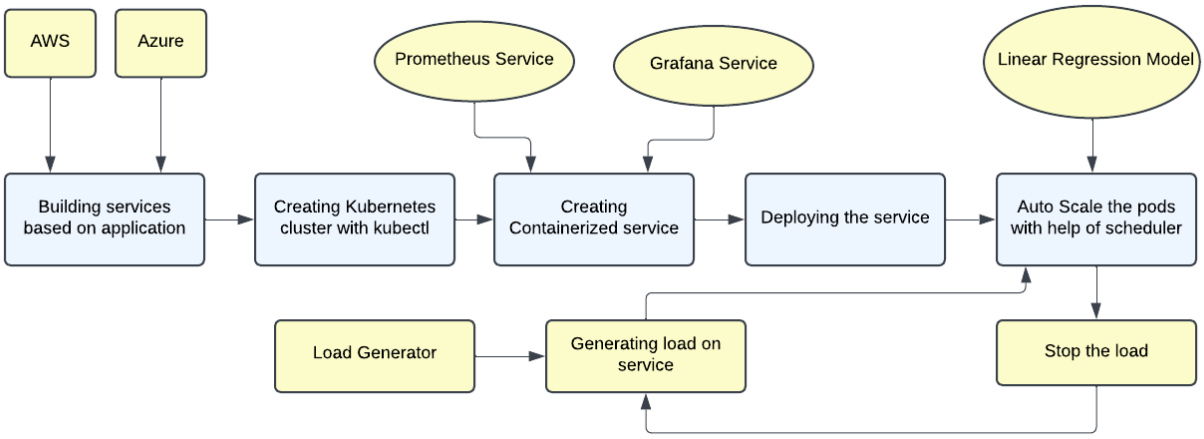

Figure 2: Proposed Workflow of research

The above workflow shown in Figure 2 can be proposed to start with development of cloud application services on AWS and Azure. We then use it to generate a Kubernetes cluster for the orchestration of containerised applications for example, prometheus for monitoring and grafana for data visualization. The containerized service runs on Kubernetes cluster, and the auto-scaling based on linear regression model is used to optimize and adjust the number of pods. A load generator is used to load the service and the load is sometimes taken off the service. This work flow is showing like how to achieve a process of deploying the application in a scalable manner in a container environment having monitoring and data analysing factors for proficient usage of resources as well as dynamism to the varying loads.

Table 2: Technologies and Tools Used in the Research

Category	Technologies/Tools
Cloud Services for AWS	Elastic Kubernetes Service (EKS) and Elastic Container Registry (ECR)
Cloud Services for Azure	Azure Kubernetes Service (AKS) and Azure Container Registry (ACR)
Programming Language	Python
Frameworks	Flask
Libraries	Numpy
Containerization	Docker
Monitoring	Prometheus
Visualization	Grafana
Custom Management	Custom Controller

Deployment Tools	Docker Commands, Kubernetes (kubectl)
Development Environment	Visual Studio Code (VS Code)

4. Design Specification

4.1 Required System Specifications

In this study, container deployment and operation are managed by orchestration services, specifically Elastic Kubernetes Service (EKS) on AWS and Azure Kubernetes Service (AKS) on Azure. Both platforms utilize Kubernetes to manage containers within the clusters, ensuring efficient scaling, load balancing, and resource allocation. Docker containers are employed to encapsulate the application code along with its dependencies, providing a consistent and portable environment for execution. By using Kubernetes and Docker, the system ensures seamless container orchestration, optimizing performance, scalability, and reliability across both cloud platforms, AWS and Azure.

Architecture Diagram:

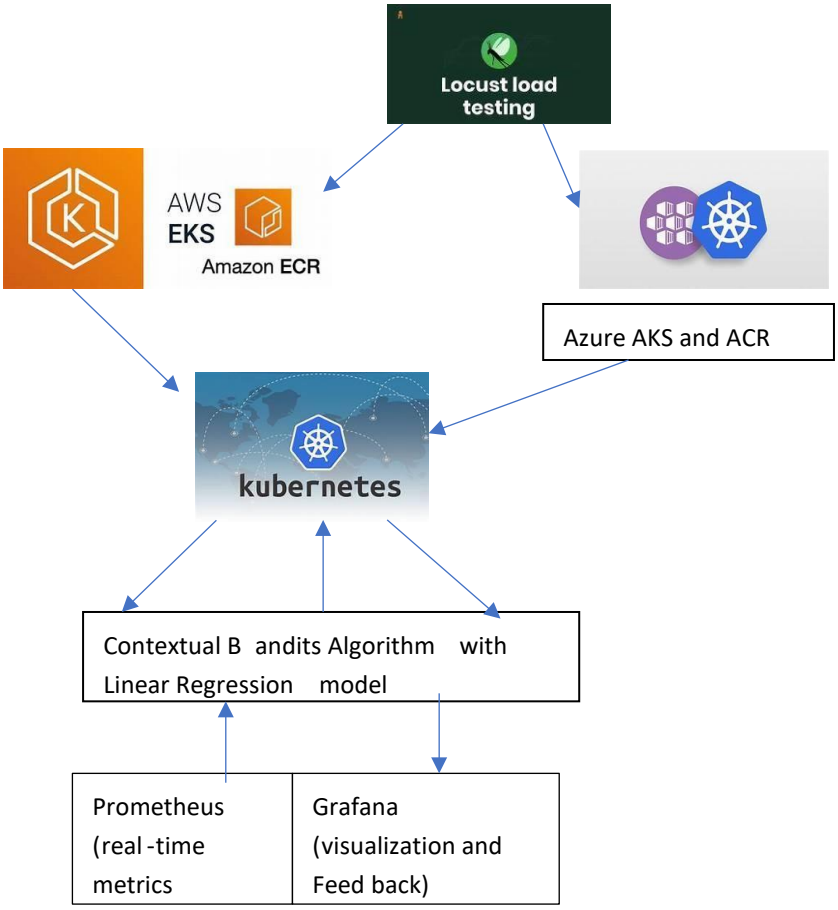

Table 3: Required System Specifications

Category	AWS Specifications	Azure Specifications
Cloud Service	Amazon Web Services (AWS)	Microsoft Azure
Instance Type	AWS t3.medium	Azure Standard_DS3_v2
Operating System	Ubuntu 22.04 LTS (Latest Stable Version)	Ubuntu 22.04 LTS (Latest Stable Version)
Kubernetes Version	Kubernetes 1.25 (Latest Stable Version)	Kubernetes 1.25 (Latest Stable Version)
Container Engine	Docker 20.10 (Latest Stable Version)	Docker 20.10 (Latest Stable Version)
Orchestration Tool	Kubernetes 1.25 (Latest Stable Version)	Azure Kubernetes Service (AKS) - Kubernetes 1.25

}

5. Implementation

5.1 Implementation of Linear Regression Algorithm

Python was used in the implementation of linear regression algorithm for this study, and the flask web application framework was used to structure the web application. The basic operations of the linear regression model were done with the help of a python library known as numpy, which is specially designed for handling of large mathematical calculations. As it has been claimed, this model was intended for outcome forecast with linear dependency on the proposed input variables and the target variable. The implementation used a build, measure, learn cycle where the first step was data preparation where normalization and management of missing values were done. When the data was prepared the numpy library was used to calculate the coefficients of the linear regression model using the Ordinary Least Squares method which provided the best fit for this model. After that, several datasets were applied in the test to ensure its usability in prediction as well as confidence of the model. The whole application such as the implementation of linear regression was containerized with Docker used. This approach helped in packaging all the code as well as its dependencies in a single Docker capable image so that we can use it in any environment. Then the Docker container was deployed to an EKS and AKS cluster which could optimally scale in response to various computational heavy demands. Moreover, for the proper cycle of functioning of the application, a customized controller was created for proper control of the system flow. Prometheus was incorporated into the system to enable real-time average CPU usage and memory consumption measurements, all of which were visualized with grafana, which can present these values on dynamic, interactive dashboards. This offered useful information on how the linear regression model performed in the real world so that the usage of resources in the production system could be closely supervised and improved where necessary. These technologies made it possible to deploy this linear regression algorithm in this study in a very robust, efficient, scalable manner.

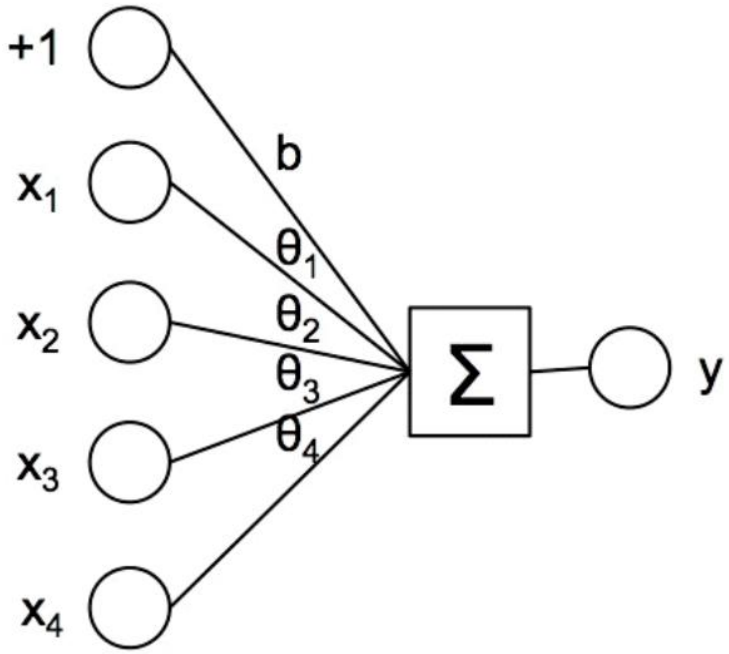

Figure 3: Architecture of Linear Regression by (Sarkar and Kashikar, 2023)

Algorithm: Linear Regression

Require: Dataset (X, y), Learning Rate (α), Epochs (E)

Ensure: Optimized Weights (θ)

1: Initialize θ, b to zero

2: for each epoch in 1 to E do

3: predictions ← X.dot(θ) + b

4: cost ← (1/m) * Σ (predictions - y) ²

5: gradients ← (1/m) * X.T.dot (predictions - y)

6: θ ← θ - α * gradients

7: b ← b - α * Σ (predictions - y) / m

8: end for

9: return θ, b

5.1 Cloud Deployment on Azure Using Docker

As indicated by figure 4, application deployment using docker alongside ACR appears on the Azure portal. The image illustrates the “cpuloadgenflask” the repository location within the “cpuloadgenacr” container registry that forms part of the cloud deployment environment. This information about the deployment is contained in the repository details. The last concept called the ‘latest’ image tag puts the latest version of the application and its related metadata like the digest – it is a unique handle to the image, last modified. This information helps to update the state of the deployed application by making a backup snapshot which can be rolled back to if necessary. From the navigation pane, a single customizable representation of the repository is available through the Azure portal interface to control features such as managing deleted artifacts, access control, health, and utilization of the container registry. At the same time, the given model helps to organize the centralized management of a containerized application, which leads to the provision of a highly reliable and scalable cloud infrastructure.

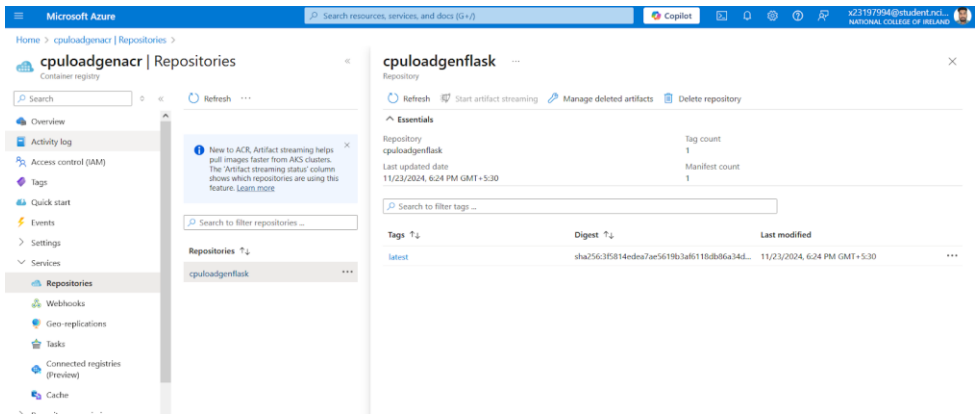

Figure 4: Cloud Deployment on Azure Using Docker and ACR (Azure Container Registry)

Figure 5 shows how the Azure portal looks for node pools as part of an Azure Kubernetes Service (AKS) cluster. Node pools in this context depict the virtual machines; that host the application workloads, in the given Kubernetes architecture model. The image shows a pattern similar to a table with sections that indicate the node pools configured for the AKS cluster. In the table, each row refers to a node pool and gives all the specification, including all related resources and their usage. Its field “Node” contains exact names of the node pools, which helps to find and work with particular node groups in a proper way. The field “Status” shows the state of nodes, so the administrator can immediately notice the state of readiness for container deployment. Moreover, in the “Pods” column, each node pool describes the number of pods being run, and “Kubernetes version” column indicates the Kubernetes version to help with version control and update.

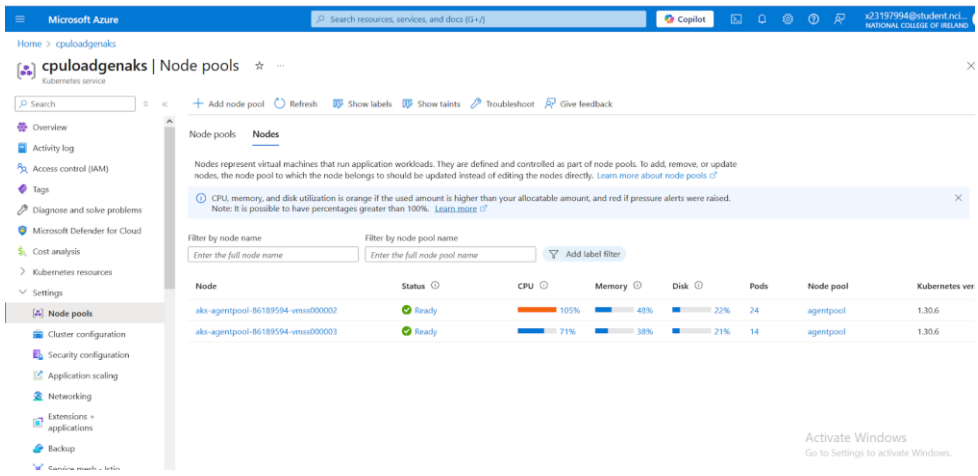

Figure 5: AKS (Azure Kubernetes Service)

5.2 Cloud Deployment on AWS Using Docker

Figure 6 illustrates the user interfaces of the Amazon ECR in the AWS Management console in terms of the specification and details of a docker image hosted in the registry. The figure highlights the view of the deployed artifact to give details and metadata of the particular

artifact. The “Details” option displays the key information about the image: the tags, the digest, which is the image’s unique identifier when it is uploaded and the URI for calling the image. The "General information" section adds even more value with regard to the deployment, identifying the artifact type as an “Image,” repository as “cpuloadgenacr,” and the time that image was registered. In the figure above, one of the most valuable features presented under the figure is called “Scanning and vulnerabilities” which gives information of the security level of the image being deployed. It shows if present day threats have been identified, and thus helping the administrator prevent or mitigate a potential risk to the security of the container-based application and its general soundness.

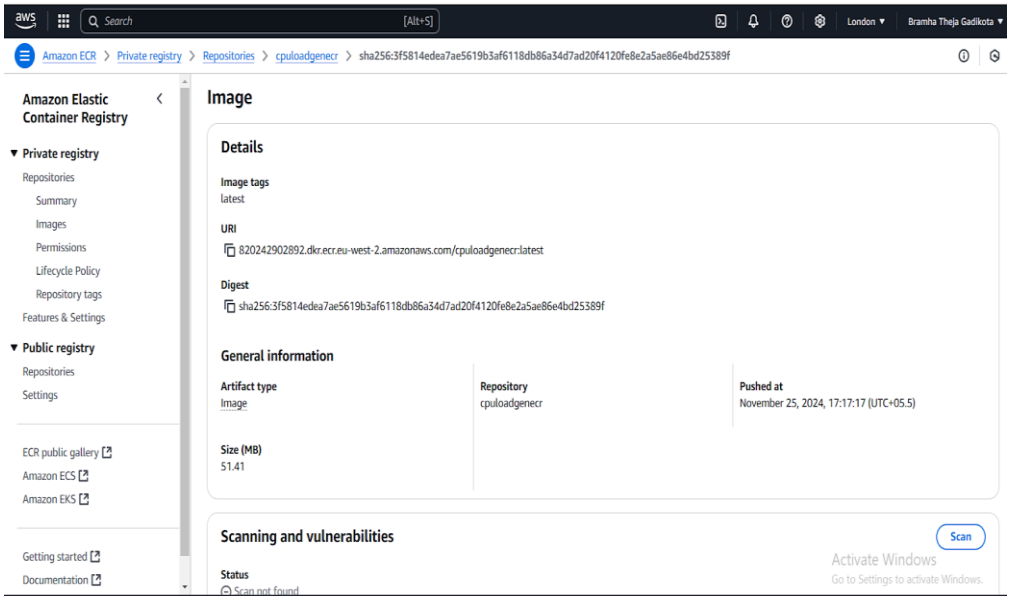

Figure 6: Cloud Deployment on AWS Using Docker and ECR (Amazon Elastic Container Registry)

Figure 7 illustrates the interface of Amazon ECR within the AWS Management Console, the screenshot is evidencing the details of a docker image in the registry. It offers information on the artifact deployed, its basic information that includes the metadata. In the present case, the “Details” section of the component brings into focus the pertinent details of the image like the image tags, the digest identifier as well as the URI for referencing the image. This is so because by deploying the cloud through the AWS ECR the whole process is made more secure and efficient. The container registry centralizes Docker image management; that allows implementing versioning, access control, to integrate with other AWS services like ECS or EKS for containerized applications management.

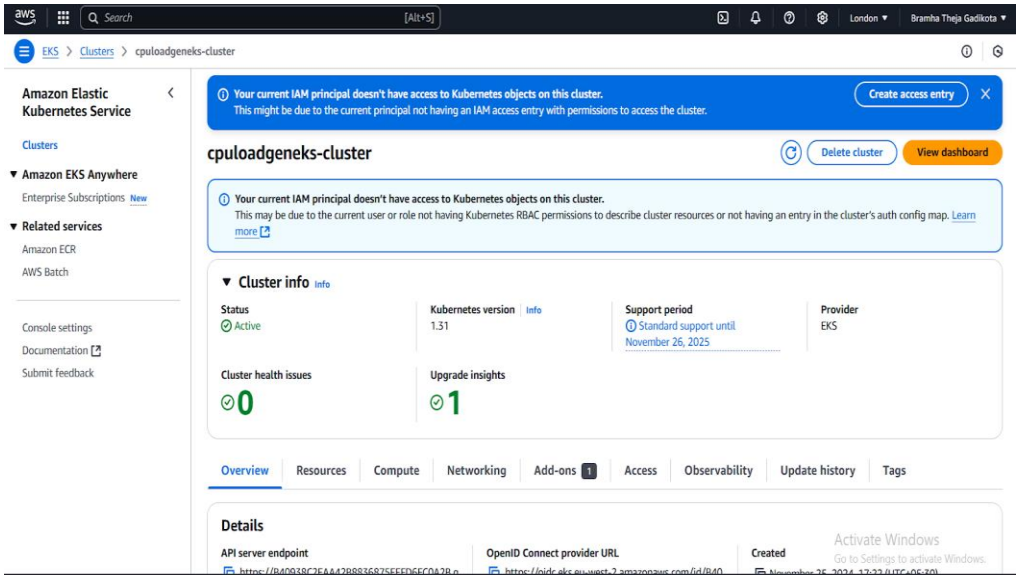

Figure 7: EKS (Amazon Elastic Kubernetes Service)

6. Evaluation

6.1 Case Study 1: AWS load testing

The load testing using the locust platform is a real-life case study that can be used to explain the use of a distributed load testing framework for measuring the performance and scalability of an application hosted in Amazon web services. The graph shows that the system was initially stable and handles the increasing load and user numbers quite well whereas the number of users are also increasing as load increases and the failure rate is also low. However, as the load increased at a certain time the response and the efficiency of the system began to decline. The failure rate went up high, which showed that the system was not capable of handling the number of requests tendered to it. The response times also soared high, indicating there was an inherent delay as the system was struggling. This goes to show that the ability of the system was exceeded, it could not accommodate the growth hence missing the point of expansion.

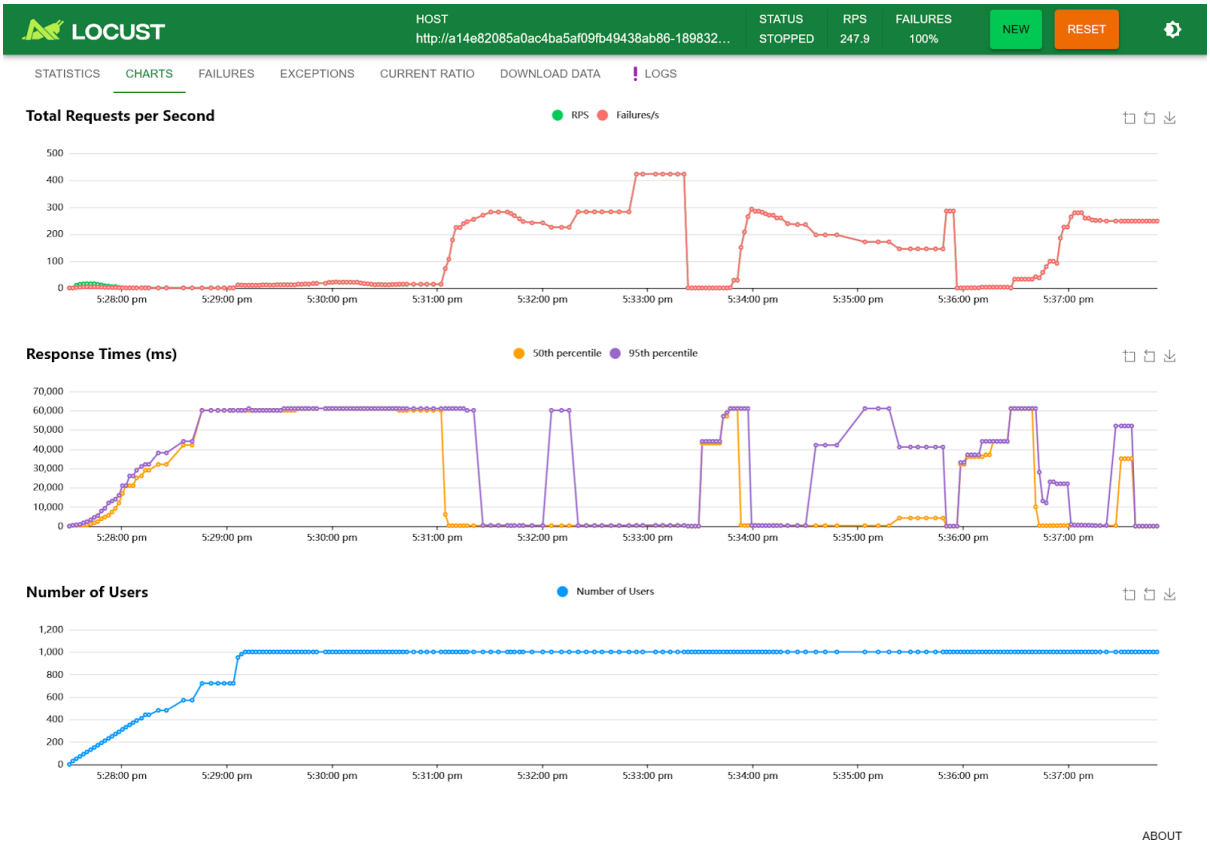

Figure 8: AWS Load Testing Case Study

6.2 Case Study 2: Azure load testing

This study presents load testing of Azure using locust as a case study to show how a distributed load testing framework can be applied in testing an application deployed on the Microsoft Azure cloud platform. The total requests per second graph contains information on the RPS and failure. Initially, the RPS starts in the range of 15-20 with a low failure rate which represents that the system is well capable of managing the load. However, as the load increases with time, the RPS increases sharply up to a point of about 22 requests per second. At the same time, there is a rapid worsening of the failure rate, which indicates that the system is too loaded and can no longer deal with the requests coming in.

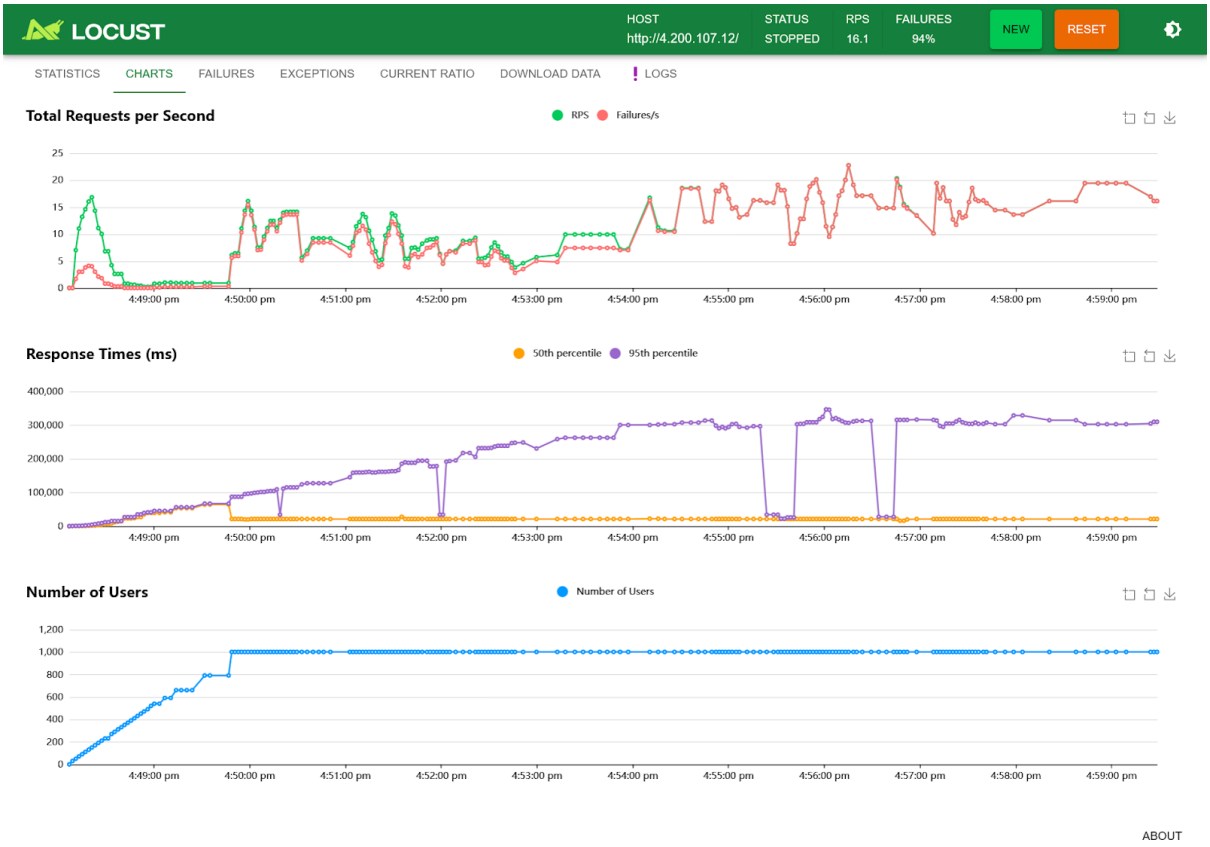

Figure 9: Azure Load Testing Case Study

Table: Comparison Table

Aspect	AWS	Azure
Platform Utilized	Amazon Web Services (AWS)	Microsoft Azure
Requests Per Second (RPS)	**247.9**	**16.1**
Failure Rate	**100%**	**94%**

6.3 Case Study 3: Experimentation of Load Testing on AWS and Azure

6.3.1 Comparison of CPU Utilization Between AWS and Azure Infrastructures

The below Figures represent CPU usage comparison of AWS and Azure infrastructure. Figure 10 displays the CPU utilization percentage for the AWS infrastructure, which fluctuates significantly over time, reaching a peak of around 75%. On the other hand, we see more consistent picture of the CPU load for the Azure infrastructure graphically presented in Figure 11 having mostly fluctuated between the 40 and 80 percent mark within the entirety of the considered timeframe. The characteristics of the two cloud providers are revealed through understanding their approach to handling and allocation of resources as well as the scalability. The trends enjoy high variance between the spikes and valleys of CPU use which may suggest a more reactive structure of AWS. On the other hand, from statistics of the Azure infrastructure, the CPU usage is much more consistent and calculated, thus meaning that the resource management is very well planned.

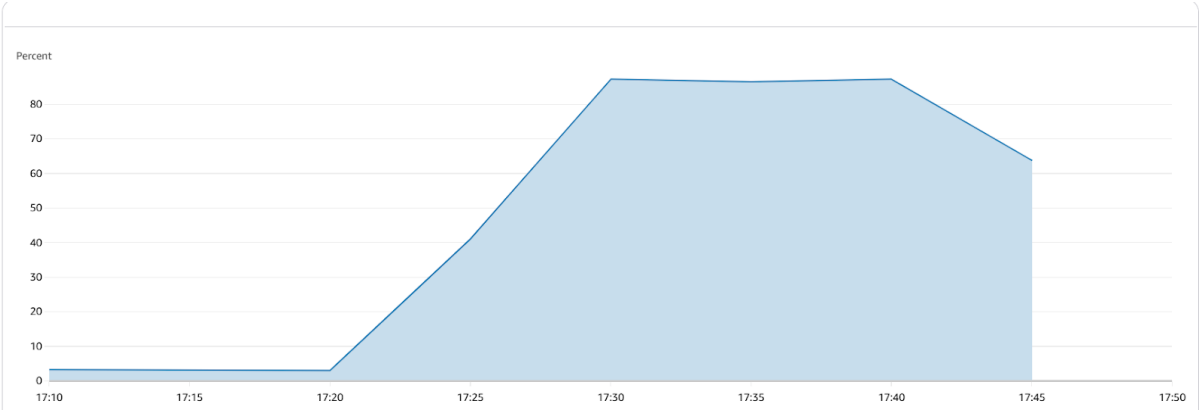

Figure 10: AWS CPU Utilization (%)

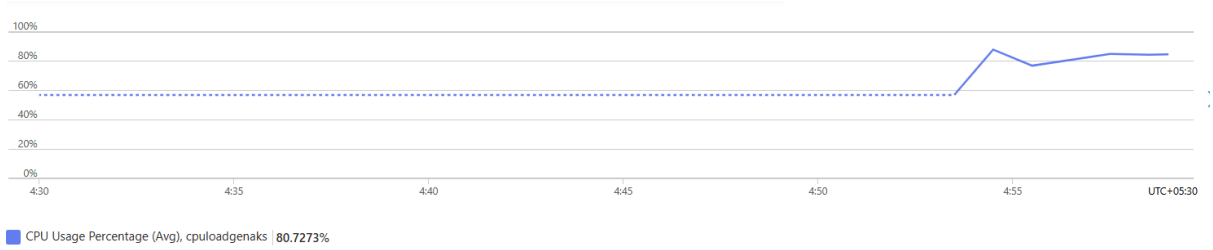

Figure 11: Azure CPU Utilization (%)

6.3.2 Comparison of Network (in Bytes) Between AWS and Azure Infrastructures

The images below compare the network traffic analysis traffic in the types of network inbound bytes in AWS and Azure infrastructures. Figure 12 depicts the network inbound traffic for the AWS infrastructure, which exhibits a significant spike around the 17:25 time mark peak to over 3GB. This implies that the AWS infrastructure activities involved in the network went higher at one point, probably due to data processing and or communication with the outside world. On the other hand, the figure below presents the network inbound traffic of Azure structures showing that it has a stable, steady pattern in contrast to frequent fluctuation. The network traffic does not rise significantly it stays in the range of 2MB-6MB during the time shown under the network column. In this case, the two cloud providers show differences in network traffic which can be suggestive of the divergence in workload characteristics, allocation of resources and most importantly the networking services forming the cloud.

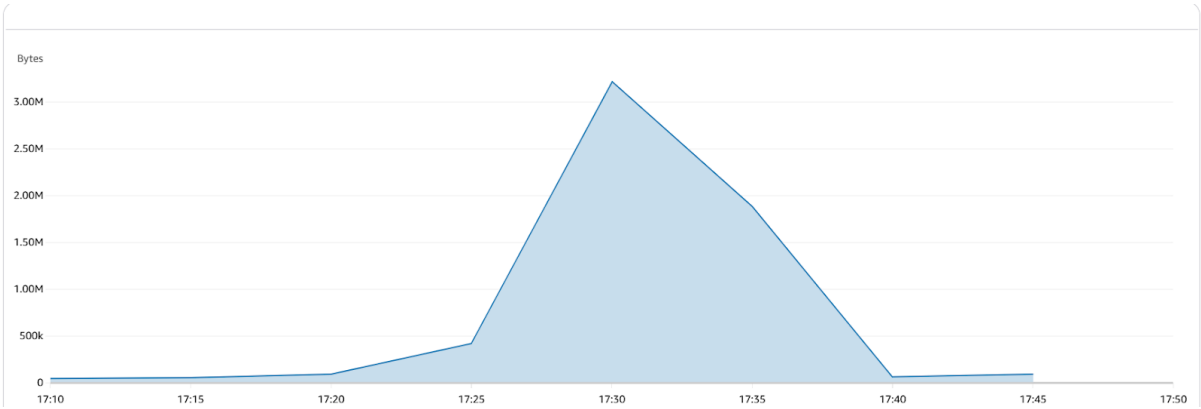

Figure 12: AWS Network In (Bytes)

Figure 13: Azure Network In (Bytes)

6.3.3 Comparison of Network (Out Bytes) Between AWS and Azure Infrastructures

The following images illustrate the network outbound traffic for the AWS and Azure infrastructures entered in this study. Figure 14 shows the network outbound bytes for the AWS infrastructure, which exhibits a significant spike around the 17:35 time mark. Out bound traffic increases with a maximum of 600k bytes, meaning that at some given instance there is increased traffic out of the AWS environment. This was most probably caused by surge on data transfer/communication on the AWS infrastructure with external services during this period. On the other hand, the Azure network outbound traffic has shown more consistent and more stable in figure 15. While there is a noticeable spike, reaching 100k bytes, overall, the Azure total out traffic stays quite low but with some spikes, it mostly alternating between 40k to 80k bytes with an average of 69.5k bytes. The dissimilarities in cases of network outbound traffic revealed slight variations in the workloads, resource utilization, and network preferences between the two leading cloud providers. These observations might suggest that the network usage for the AWS environment is more bursty, or has more daily fluctuations than the relatively stable showing of the Azure infrastructure.

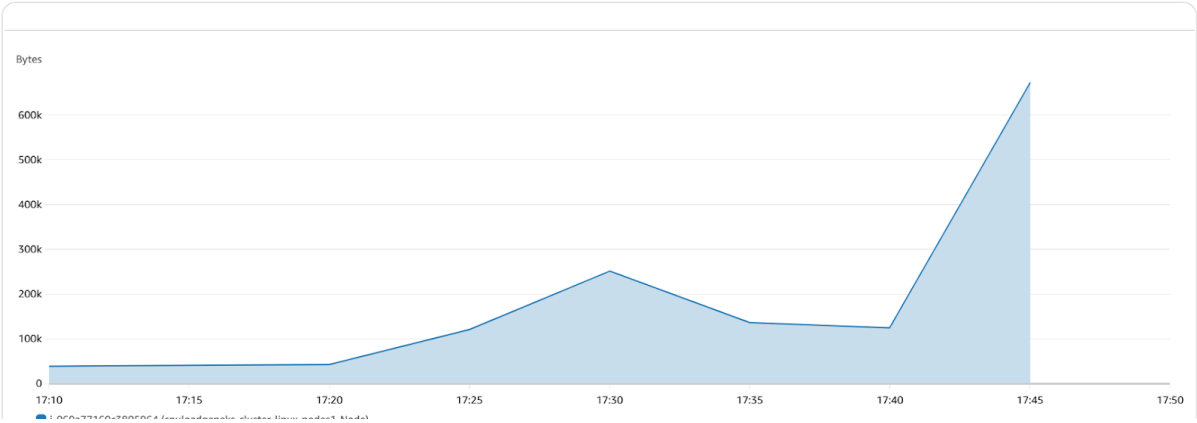

Figure 14: AWS Network Out (Bytes)

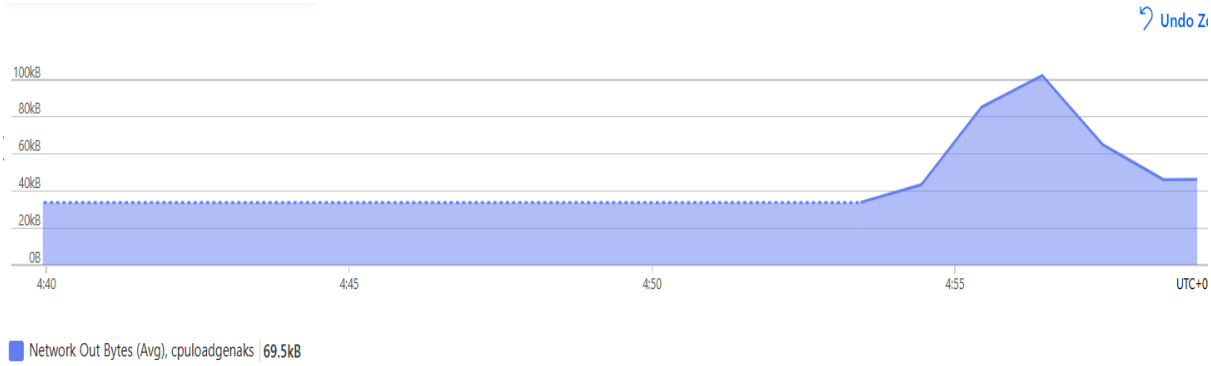

Figure 15: Azure Network Out (Bytes)

6.3.4 AZURE node CPU & node memory utilization

From the above comparisons, it is understood that over resource utilization of Azure outperforms AWS. Figure 16 shows the CPU and memory utilization for nodes in the Azure infrastructure over time. The CPU utilization graph displays an average utilization of 46.10%, with a maximum utilization of 99.97%. This indicates that the CPU usage on the Azure nodes fluctuates significantly, with periods of high demand that reach near-maximum capacity. The memory utilization graph, on the other hand, shows a more stable pattern. The average memory utilization is 30.34%, with a maximum of 36.03%. This suggests that the Azure nodes have sufficient memory resources to handle the workloads, with the memory usage remaining within a comfortable range throughout the period. The difference between the CPU and memory utilization patterns suggests that the Azure infrastructure may be more constrained by CPU resources than memory resources. This could be an important consideration when scaling or optimizing the infrastructure to meet changing demand.

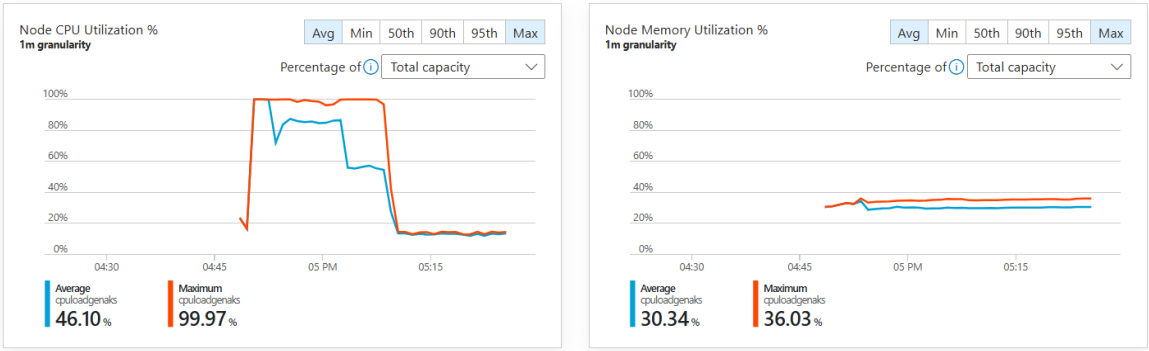

Figure 16: AZURE node CPU & node memory utilization in %

6.4 Discussion

Through the study we evaluate how the application performs between AWS and Azure infrastructure platforms when Kubernetes orchestration handles their management. The findings reveal substantial variations exist between platform scalability and reliability capabilities. The RPS performance of AWS reached 257.9 while Azure managed 16.1 but both systems encountered failures at 100% and 94% respectively. Kubernetes cluster resource allocation and scheduling algorithms encounter significant difficulties when-founded on cluster resources. This behavior becomes worse as workload intensity increases. Advanced resource scheduling methods demonstrate their effectiveness through implementation of contextual bandits combined with linear regression models. The technique finds equilibrium between detection of new opportunities and the current resource distribution to maximize utilization efficiency while maintaining stability and avoidance of system breaks. What drives the AWS RPS to higher levels is either better resource capacity configurations or improved resource availability yet the observed failure rates point to the necessity of enhancing both load management and resource scalability for effective optimization. Monitoring platforms Prometheus and Grafana provided essential capabilities for metric visualization so users could detect system bottlenecks. Although both AWS and Azure provide suitable platforms for Kubernetes-based deployment AWS delivers slightly better results yet none of these platforms operate flawlessly during periods of heavy load which highlights ongoing requirements to enhance resource performance strategies.

7. Conclusion and Future Works

7.1 Conclusion

This study was able to show how a cloud-based system using modern technology can be implemented and deployed to effectively schedule, monitor and optimize resources. Using two major cloud-based solutions, Amazon Web Services (AWS) and Microsoft Azure, key cloud platforms, the system was stressed tested to simulate load conditions. Used code for linear regression brought out the prospect of applying machine learning in resource management and the best/next-use problems. Containerization using Docker in combination with Kubernetes allowed for proper resources management and unproblematic applications' deployment. Prometheus and Grafana were well capable in monitoring aspects such as CPU

and memory utilization which were important evaluating the performance of the system. Stress testing the system with Locust provided information about the performance of the application under increasing loads of users; understanding system constraints was crucial. As the analysis revealed AWS achieved higher RPS but with more failure rate as compared to Azure showing a clear correlation enhanced for future practical application. In aggregate, the project emphasized the importance of cloud technologies, monitoring tools and load testing frameworks as the key instruments for creating effective and fully-scaled applications with high performance.

7.2 Future Works

Further work can be directed towards improving the stability, expandability and resource utilisation of the computing system. The use of contextual bandits in machine learning to make dynamic decisions regarding the resources in Kubernetes will lead to better decisions and better performance. Including elevation of AWS and Azure resources into one network infrastructure, and thereby it could increase the level of redundancy and flexibility for the company. Adding more advanced monitoring tools of system usage and probable failure with AI may help to more efficiently utilize the resources. The integration of edge computing with cloud resource could decrease latency for real time application. Further improvement to load testing is possible by emulating more complicated user interactions and incorporating fault injection tests which yield a better understanding of application performance under pressure. Moreover, lots of updates can be easily implemented with CI/CD tools of the deployment pipeline and would minimize downtime. In order to increase failure rates, particularly on AWS, future measures, including load balancers, autoscaling and database indexes, must be examined. For Azure, the future enhancements could be in form of moderating latency and enhancing RPS through better caching. Last but not the least, the main model can extend future work by adopting sustainability measures such as carbon-aware scheduling of resources to promote environmentally sustainable practices without compromising performance or scalability.

References

1. Bietti, A., Agarwal, A. and Langford, J., 2021. A contextual bandit bake-off. *Journal of Machine Learning Research*, *22*(133), pp.1-49.
2. Bouneffouf, D., Rish, I. and Aggarwal, C., 2020, July. Survey on applications of multi-armed and contextual bandits. In *2020 IEEE Congress on Evolutionary Computation (CEC)* (pp. 1-8). IEEE.
3. Carrión, C., 2022. Kubernetes scheduling: Taxonomy, ongoing issues and challenges. *ACM Computing Surveys*, *55*(7), pp.1-37.
4. Casalicchio, E. and Iannucci, S., 2020. The state-of-the-art in container technologies: Application, orchestration and security. *Concurrency and Computation: Practice and Experience*, *32*(17), p.e5668.
5. Førre, M.R., 2022. *Beyond bin-packing combining bin-packing algorithms tailored for automated software testing: ROST-algorithm* (Master's thesis, OsloMet-storbyuniversitetet).
6. Hanna, O.A., Yang, L. and Fragouli, C., 2023, July. Contexts can be cheap: Solving stochastic contextual bandits with linear bandit algorithms. In *The Thirty Sixth Annual Conference on Learning Theory* (pp. 1791-1821). PMLR.
7. Islam, M.S.U., Kumar, A. and Hu, Y.C., 2021. Context-aware scheduling in Fog computing: A survey, taxonomy, challenges and future directions. *Journal of Network and Computer Applications*, *180*, p.103008.
8. Lövenvald, F.L., 2021. Comparing a gang-like scheduler with the default Kubernetes scheduler in a multi-tenant serverless distributed deep learning training environment.
9. Menouer, T., 2021. KCSS: Kubernetes container scheduling strategy. *The Journal of Supercomputing*, *77*(5), pp.4267-4293.

10. Miller, S., Siems, T. and Debroy, V., 2021, October. Kubernetes for cloud container orchestration versus containers as a service (caas): practical insights. In *2021 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)* (pp. 407-408). IEEE.
11. Praveenchandar, J. and Tamilarasi, A., 2021. Retracted article: dynamic resource allocation with optimized task scheduling and improved power management in cloud computing. *Journal of Ambient Intelligence and Humanized Computing*, *12*(3), pp.4147-4159.
12. Qunaibi, S., 2023. *Improving Cluster Scheduling Resiliency to Network Faults* (Master's thesis, University of Waterloo).
13. Rejiba, Z. and Chamanara, J., 2022. Custom scheduling in Kubernetes: A survey on common problems and solution approaches. *ACM Computing Surveys*, *55*(7), pp.1-37.
14. Sarkar, A. and Kashikar, P., LITERATURE REVIEW OF IMPLEMENTATION OF MACHINE LEARNING ALGORITHMS FOR IMPROVING THE NETWORK SECURITY.
15. Schumann, M., 2024. *Conceptual design of a container-based system landscape orchestrated by Kubernetes* (Bachelor's thesis).
16. Vaño, R., Lacalle, I., Sowiński, P., S-Julián, R. and Palau, C.E., 2023. Cloud-native workload orchestration at the edge: A deployment review and future directions. *Sensors*, *23*(4), p.2215.