National College of Ireland

Optimizing Workload Scheduling And Cost Efficiency Using Cloud Orchestration Frameworks In Multi-Cloud Environments

MSC Research Project
Cloud Computing

Rushikesh Sawant
Student ID: 23303891

School of Computing
National College of Ireland

Supervisor: Sean Heeney

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Rushikesh Sawant
Student ID:	23303891
Programme:	Cloud Computing
Year:	2024 - 2025
Module:	MSC Research Project
Supervisor:	Sean Heeney
Submission Due Date:	11/08/2025
Project Title:	Optimizing Workload Scheduling and Cost Efficiency Using Cloud Orchestration Frameworks in Multi-Cloud Environments
Word Count:	9418
Page Count:	26

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:	Rushikesh Sawant
Date:	11th August 2025

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):	

Optimizing Workload Scheduling and Cost Efficiency Using Cloud Orchestration Frameworks in Multi-Cloud Environments

Rushikesh Sawant
Student ID:23303891

Abstract

The investigation involves an empirical analysis of workload schedules and the topics of cost optimisation and cost comparison in Kubernetes-based multi-cloud environments that utilise Amazon Web Services (AWS), Microsoft Azure, IBM Cloud, and Google Cloud Platform (GCP). The paper examines this by looking at the benefits of using real-time metrics (using Datadog to monitor performance, Big Query to analyse costs, and Google Sheets to display the cost metrics in real time visually) to make decisions regarding where to place workloads across providers. The same workloads were run in a multi-cloud scale setting, and the results measured latency, availability, throughput, cost and the complexity of operations. The findings show that comparison between multiple cloud providers shows large savings in costs and resiliency through dynamic elastic exploitation of provider-specific cost advantages and performance differences through workloads, and single-cloud deployments provide simplified operational management and predictable behaviour on stable workloads. Nonetheless, the multi-cloud orchestration complexity is key to the need to have powerful data integration and observability frameworks. The paper has delivered benchmarks, cost-performance trade-off analysis, and useful recommendations to schedule workloads in Kubernetes, making it operational and of use to the cloud practitioners in having actionable advice to inform data-driven deployments.

1 Introduction

In the world of computing, Cloud computing plays a major role. It has emerged as a fundamental enabler of scalable, flexible and cost-effective IT infrastructure. As many organizations are moving towards a hybrid cloud rather than relying on any single cloud, the complexity of managing resources efficiently across multiple environments has become the central challenge. Also selecting which cloud provides the best performance by keeping cost as a condition, The multi cloud strategies involve leveraging services from multiple cloud vendors like AWS, AZURE, GCP, IBM etc are some of the popular vendors, for their potential to improve reliability, avoid vendor lock in and optimize performance, however all these constraints by keeping cost in mind. It also comes with another challenge: deploying applications across multiple cloud platforms introduces new sets of challenges, such as workload management, cost tracking, and service interoperability are some of the challenges to overcome this Kubernetes has been introduced as a powerful tool for managing containerised applications, offering orchestration capabilities that streamline deployment, scaling and operations. Kubernetes was designed for single-cloud environments, and it lacks native capabilities to dynamically schedule workloads based upon metrics such as real-time cost, latency and availability across the providers.

The main motivation behind this research is the growing need for organizations, especially teams like DevOps, Cloud Architects, to make supported and data-driven decisions when orchestrating applications in a multi-cloud setup. The benefits of multi-cloud have been widely recognized theoretically, but there is a lack of practical tools and frameworks that provide real-time insights into cost and performance across different cloud vendors. This can lead to issues like resource allocation, high expenses and can also cause unnecessary latency and downtime. Most teams—particularly those that are new to multi-cloud setups—confront a significant obstacle: a dearth of consolidated systems that allow them to monitor, analyse, and act upon data in real-time. Datadog-like tools are excellent for monitoring application and system performance metrics (i.e., CPU utilization, memory, response times). Still, these metrics by themselves are insufficient for making completely informed scheduling choices when you are executing workloads on various cloud providers (e.g., AWS, GCP, or Azure).
What is missing is the capacity to bring together performance data, cost data, and operational log data in real time. Datadog, for example, can tell you that a VM is under high load, but it won't tell you whether offloading that work to another cloud provider will be more or less expensive. BigQuery is a product that can help analyse and query cost data in real time, and Google Sheets can be used for visualizing this data quickly and presenting it to decision-makers.

2 Research Question

How can comparing costs and performance of different cloud providers help schedule workloads better in a Kubernetes setup that uses multiple clouds?

1. This research question addresses the issue of selecting an appropriate cloud provider based on an organisational need when deploying applications across a multi-cloud platform. As organisations are switching to a multi-cloud environment, understanding the real-time cost, performance, and latency plays a crucial role in ensuring both efficiency and reliability.

2. This work is significant as it ties together concepts of cloud services management and operational data. Cloud services vary a lot depending on the region of use, the services offered, and cost. If we do not consider cost and performance, the work shall not be performed well and thus incur higher cloud costs, poor user experience, or underutilised resources.

3. The core research question, the study is guided by the following objectives: Put AWS, Azure, IBM and GCP to the test. Note the speed, accessibility, processing of data and cost for the same workload deployed on Kubernetes.

To see performance data, utilize live monitoring tools such as Datadog, and access billing and system logs with BigQuery. Examine the performance and cost variations across different geographies and load conditions. Test Kubernetes services across many cloud platforms, including AWS, AZURE, IBM, and GCP, to analyze latency, availability, throughput, and cost in equivalent workload situations. Identify and use patterns and similarities that impact cost and performance amongst providers to determine the best division of labour. Provide data-driven suggestions to companies to optimise workload placement based on observed cost-performance trade-offs.
Hypothesis: Prices and performance levels vary greatly between cloud providers and locales. If we carefully investigate these variations, we may uncover the best approaches to plan that provide the most cost savings and ideal performance. This article uses actual data to provide recommendations for improving workloads in a multi-cloud context. This is accomplished without the use of modern scheduling methods. The research also analyzes the variation in performance and cost in multi-cloud systems.

3 Related Work

This section relates the work presented herein to prior work. It provides a clear overview of existing research on multi-cloud computing, workload scheduling, cost optimization, and orchestration models with Kubernetes. The purpose is to indicate significant contributions to the work, clarify the strengths and limitations of the contributions, and indicate the distinctions or improvements over prior work.

3.1 Single Cloud vs. Multi-Cloud Cost-Performance Comparisons

Tambí, V. K. (2025) – Scalable Kubernetes Workload Orchestration for Multi-Cloud Environments (TRJ, Jan–Mar 2025)

Introduces a federated Kubernetes orchestration system that schedules workloads on AWS, Azure, and GCP and optimizes various metrics like startup latency, failover time, CPU usage, and cost overhead.
Limitations: Does not incorporate real-time telemetry or expense monitoring from tools such as Datadog or BigQuery, nor does it involve systematic comparison with single-cloud deployments on the same workloads.
Gap: Does not analyse empirically cost and performance comparisons between single-cloud and multi-cloud deployment, also does not analyze the real-time performance, cost information during scheduling decisions.

ArXiv (Apr 2025) – Kubernetes in the Cloud vs. Bare Metal: A Comparative Study of Network Costs

Compares the network cost variance between classic bare-metal and cloud deployment of Kubernetes workloads.
Limitations: Focused solely on network egress cost; does not include multi-cloud performance metrics, scheduling decisions, or comparative analysis with single-cloud environments. Gap: Does not include multi-cloud comparisons and multi-dimensional cost-performance tradeoffs; restricted to network-layer analysis.

3.2 Cost-Aware & Multi-Cloud Scheduling Techniques

Zambianco, M., Cretti, S., & Siracusa, D. (2024) – Cost Minimization in Multi-cloud Systems with Runtime Microservice Re-orchestration, ICIN 2024

Proposes an ILP-based rolling update method to re-orchestrate microservices among cloud providers in real time, with minimal cost while maintaining QoS. Results beat Kubernetes scheduler baselines.
Limitations: Built for microservice-level control instead of Kubernetes pods, no integration with live telemetry streams, and deployment under production-like loads.
Gap: The lack of actual real-time telemetry integration (Datadog, BigQuery) and Kubernetes-native scalable assessment of workloads across clouds.

Edirisinghe, D., Rajapakse, K., Abeysinghe, P., & Rathnayake, S. (2024) – SpotKube: Cost-Optimal Microservices Deployment with Cluster Autoscaling and Spot Pricing

Describes a genetic algorithm scheduler based on Kubernetes that utilises spot-priced nodes and dynamic autoscaling for cost optimization while handling availability disruptions.
Limitations: Concentrates on spot-instance situations, doesn't address multi-cloud provider comparisons or include performance/latency trade-offs among providers.
Gap: Does not look at cross-provider cost-performance or multi-cloud scheduling techniques.

3.3 Intelligent and RL-Based Scheduling in Kubernetes

Xu, Z., Gong, Y., Zhou, Y., Bao, Q., & Qian, W. (2024) – Kubernetes Automated Scheduling with Deep Learning and Reinforcement Techniques, arXiv February 2024

Uses a deep learning + reinforcement learning model to adjust scheduling according to real-time resource forecasts, enhancing execution efficiency.
Limitations: Tested in simulated environments, not multi-cloud, and without integrated cost analytics or real-world monitoring tools.
Gap: Has not undergone any empirical validation within actual Kubernetes clusters and any live cost analytics across providers.

Pradeep, P. & Al-Masri, E. (2025) – Energy-Optimised Scheduling for an IoT Workloads Using TOPSIS, arXiv June 2025

Proposes GreenPod, an energy-efficient and performance-aware scheduler relying on multicriteria decision analysis for Kubernetes clusters.
Limitations: Emphasis on energy efficiency in homogeneous clusters—no multi-cloud setting, costperformance trade-off, or real-time analytics.
Gap: Does not consider cost and performance scheduling across multiple clouds and is not integrated with real time telemetry.

3.4 Hybrid and Security-Aware Multi-Cloud Analysis

Polinati, A. K. (2025) – Hybrid Cloud Security: Balancing Performance, Cost, and Compliance in Multi-Cloud Deployments, ResearchGate case-study analysis describes hybridcloud deployments (AWS + Azure) with a focus on cost, performance, compliance, and security.

Provides high-level framework guidance.
Limitations: More conceptual and in need of greater empirical cost-performance benchmarking; no Kubernetes-specific deployment scenarios or real-time integration.
Gap: Lacks cost-performance benchmarking as well as the results of real-life Kubernetes implementations.

3.5 Contribution of This Study

Directly comparable cost, latency, and performance metrics between single-cloud and multi-cloud Kubernetes deployments on the same workloads.Integration of real-time telemetry tools—Datadog for performance, BigQuery for billing analytics, and Google Sheets for visualization. Empirical stress testing on AWS, GCP, and Azure to determine benchmarks and trade-offs. Data-driven guidelines for practitioners to make placement decisions that are informed by quantifiable cost-performance distinctions—without complete dependence on predictive or heuristic models.

3.6 Summary and Gaps

The literature reviewed demonstrates continued research efforts in enhancing cost performance optimization in cloud computing, specifically through scheduling frameworks, orchestration tools, and deployment strategies in single-cloud and multi-cloud deployments. Single-cloud deployments, though enjoying centralized control and optimized performance tuning, tend to suffer from vendor lock-in, regional restrictions, and varying pricing models.

Multi-cloud deployments, however, provide better resilience, flexibility, and cost variety but add complexity in orchestration, performance visibility, and real-time cost monitoring.

Emerging orchestration platforms such as Kubernetes are at the heart of this transformation, allowing for application portability and dynamic scaling. Yet, the default Kubernetes scheduler cannot optimize workload placement according to real-time cost and performance metrics from various cloud providers. Recent research has brought in improvements such as ILP-based schedulers, reinforcement learning-based agents, and edgecloud-aware plugins—but they usually miss empirical multi-cloud comparisons, integration of real-time cost data, or full-stack observability. This research fills these essential gaps by:

Study	Focus	Key findings	Strengths	Weaknesses	Gap addressed by this study
Zambianco et al. (2024)	Multi-cloud microservice re-orchestration	ILP model minimizes cost with low QoS disruption	Algorithmic optimization	Not Kubernetes-native; lacks real workloads	Empirical Kubernetes based multicloud comparison
Edirisinghe et al. (2024)	Spot-based cost optimization in K8s	Uses GA to optimize for cost and autoscaling	Cost-aware scheduler	No cross provider analysis	Comparing cost trade offs across pricing models and providers
Xu et al. (2024)	DL & RL for Kubernetes scheduling	Improves efficiency with adaptive scheduling	Predictive performance	Simulated environment only	Real-time metric based live orchestration
Tambí et al. (2025)	Multi-cloud orchestration (federated)	Highlights failover and startup latency differences	Federated Kubernetes orchestration	No telemetry or cost tracking	Integrating monitoring and cost analytics into orchestration
Madupati et al. (2025)	Multi-cloud architecture and security	Reviews architectural and governance considerations	Holistic platformlevel view	Conceptual only, lacks empirical data	Providing empirical costperformance analysis of deployment models

ArXiv (2025)	Cloud vs. bare-metal network cost	Identifies network egress as hidden cost factor	Network-layer benchmarking	Focus only on network; no scheduling view	Exploring network and compute cost together in workload analysis
Polinati et al. (2025)	Hybrid cloud deployment strategies	Proposes high-level policy model balancing compliance	Security performance cost trade-offs	No metrics or tooling involved	Evaluating orchestration impact on hybrid workload distribution

Table 1: Summarization of related works

Conducting empirical, real-time comparisons of AWS, GCP, IBM and Azure with the same Kubernetes-deployed workloads. Recording cost, latency, and performance metrics with Datadog, BigQuery, and Google Sheets. Assessing single-cloud and multi-cloud deployment results for real performance trade-offs and cost considerations. Offering actionable, data-oriented advice to DevOps professionals for optimizing workloads among cloud vendors.

4 Research Methodology

Building on the background set in the related work, this paper contributes a detailed, empirical comparison of four major managed Kubernetes services—Amazon EKS, Google GKE, Microsoft AKS, and IBM Cloud Kubernetes Service (IKS)—with a specific focus on latency, scalability, workload behaviour, and cost optimization.

4.1 Research Objective

The goal is to measure and directly compare the performance and cost-effectiveness of each platform under varying traffic conditions using a standard application stack. Through conducting controlled load tests with multiple providers, this study aims to identify the optimal deployment strategy from both a performance and economic perspective.

4.2 Experimental Setup

Application Stack: A containerised Flask API backed by a PostgreSQL database is the central test application. Docker is utilised in packaging as well as deploying the application, and images are stored on Docker Hub or Google Container Registry according to platform compatibility.

4.3 Cloud Clusters

Amazon EKS, Google GKE, Microsoft AKS, and IBM Cloud IKS were used to provision four managed Kubernetes clusters that were geographically close, in the regions: us-east-1, europewest-1, east us and lon04, respectively. This geographic distribution has been selected

because it is supposed to introduce a similar latency baseline, minimizing the region influence on the performance indicators. Every cluster was set up with similarly-specced nodes, i.e., 2 vCPUs and 4 GB of RAM, to keep parity between providers so that any differences in performance or cost could be laid squarely at platform capability- rather than infrastructure differences.

4.4 Service Exposure

In order to accomplish external access and standardized testing, each application was exposed through a load-balancer-type Kubernetes service that was able to issue public IP addresses that could instantly redirect traffic. It was possible to test across clusters in a uniform manner with publicly accessible endpoints via this configuration. To do monitoring and observability, Datadog was deployed into all clusters, giving a real-time view of important aspects like CPU and memory utilization, container health, network traffic, and behaviors per pod. This was possible due to the extensive Kubernetes integration, positively affecting Datadog visibility of the application layer latency and performance data in correlation to the infrastructure-level telemetry during load tests in a holistic view of system health, regardless of the environment of deployment.

4.5 Load Testing Strategy

This approach to load testing was constructed on the open-source tool k6, which made it possible to simulate the specific traffic patterns on the established scenarios. Three levels of load were established, including Minimum Load 10 Virtual Users (VUs) to 1 minute; Medium Load 50 VUs sustained 5 minutes; and Heavy Load (spike and sustained stress testing with 100 to 200 VUs). These were the load profiles meant to test performance, scalability, and failure thresholds of each platform on controlled but realistic bursts of traffic simulating a normal dayto-day ride as well as edge cases.

4.6 Test Scenarios

We assigned each load level to its own customised JavaScript-based k6 test script- min_test.js, medium_test.js and max_test.js. These scripts set the ramp-up schedules, continued workloads and test times. It was executed both locally and through k6 Cloud, so it has both on-premise reproducibility as well as fancy cloud-based visualization using Grafana Cloud Dashboards. The versatility of this dual-execution model gave it the ability to validate that the results could be replicated across environments, giving them a complete perspective on how the system performs in both peak and idle conditions.

4.7 Metrics Collected

The system used in testing was able to obtain detailed metrics, in order to determine the effectiveness of the simulated traffic, which included HTTP request time (an average and a 95th percentile), Requests per Second (RPS) and the error percentages, like time-outs or errors on the server side. Also, the consumption of infrastructure resources such as CPU, memory usage, pod restarts and network throughput was monitored at a node and container level by using Datadog. This multi-layered set of metrics allowed for global analysis of application-wide performance and the usage of backend computing resources.

4.8 Cost Optimization Considerations

One of the main points in this study was the analysis of the cost-effectiveness of every cloud provider. Cost data was obtained through provider documents and billing APIs, and concentrated on per-minute and per-hour costs of nodes and load balancers, etc. Cluster rightsizing methods were discussed to deduce the best node configurations in terms of better performance and cost. Autoscaling behaviour was also evaluated, especially during bursty- or sustained-traffic scenarios. Expenses in balanced inactive conditions had been compared with active outbreaks to be aware of the overhead cost of operations, which has shaped the foundation of computing the Total Cost of Ownership (TCO) by alternative traffic and providers infrastructure.

5 Design Specification

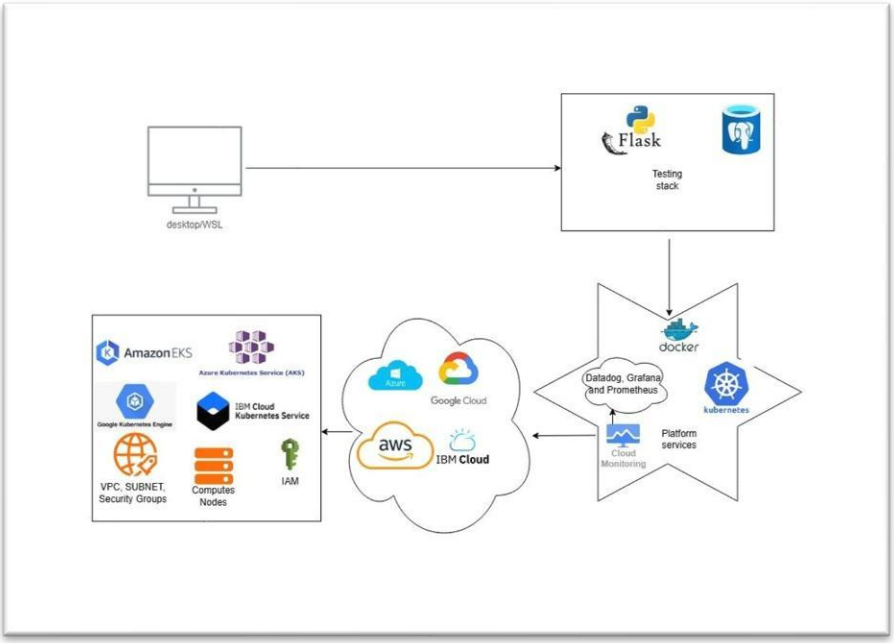


Figure 1: Architectural Diagram

In this section, the technical infrastructure and methodological design are described to perform the managed Kubernetes services evaluation in four cloud environments: Amazon EKS, Google GKE, Microsoft AKS, and IBM Cloud IKS. Each platform executed the same Flask + PostgreSQL application stack with the same number of nodes, and each cluster was publicly exposed using Load Balancers to perform load testing. The design aimed to have consistency in application behavior and infrastructure configuration to make specific comparisons in latency, scalability, cost, and observability with controlled test conditions.

5.1 Infrastructure Setup

Every cluster of Kubernetes was installed in a geographically meaningful location: EKS (useast1), GKE (europe-west1), AKS (eastus), and IKS (lon04). Each cluster ran managed

Kubernetes version 1.27-1.32, and node instances had 2 vCPUs and 4-8 GB memory (e.g., t3.medium, e2medium, Standard r2 B2s). One to two copies of an application were used per cluster. The services to provide public exposure were provided by the help of Load Balancer, and each deployment consisted of a Flask-based API container and a PostgreSQL database, which were organized (using Kubernetes manifests, i.e., deployment.yaml, service.yaml and configMap.yaml).

5.2 Applications Deployed

The application stack was a Flask API that served HTTP requests and connected to postgresql database. The PostgreSQL service was either in the same pod or otherwise a distinct deployment within the same cluster. All the services could be found internally by using DNS (mypostgres.default.svc.cluster.local). It Dockerized the Flask app and uploaded it to Docker Hub so that it can easily be reused. Equal environment setup and health checks was done on all clusters to create equitable test conditions and integrity of performance comparisons.

5.3 Monitoring and Load Testing Tools

The major load testing tool k6 was adopted to create controlled traffic in three scenarios; Minimal Load (10 VUs, 1 min), Medium Load (50 VUs, 35 min), and Heavy Load (100 200 VUs, 6 10 min). The tests have been performed both locally and using the k6 Cloud, and the visualizations in Grafana Cloud. JavaScript-based scripts were able to do this in a precise way through custom scripts. To monitor the clusters, Datadog was embedded into all of them to observe cluster health, CPU/memory usage, network traffic and HTTP-level metrics. Load testing and its phases were correlated with infrastructure use and application behavior in real time with a dashboard.

5.4 Visual Analysis & Reporting

The results of all the load tests were correlated to an integrated dashboard, where the key performance indicators (KPIs) on each of the platforms could be usefully compared side by side. Some visualizations were the latency distribution, error spike, as well as resource saturation curves, and others were the scaling behaviour timelines. These visual aids helped determine performance anomalies, infrastructure stress points and the interpretation of costefficiency trade-offs across providers- to facilitate an evidence-based final assessment of each Kubernetes service.

6 Implementation

The implementation stage of the current research implied the deployment of a common web application with the use of Flask to all four major managed Kubernetes environments, including Amazon EKS, Google GKE, Microsoft AKS, and IBM Cloud Kubernetes Service (IKS), so that their performance, cost-effectiveness, scalability, and monitoring opportunities could be compared comprehensively. This chapter describes the full deployment plan, alongside infrastructure configuration, observability tooling, performance test framework, cost monitoring and developing a prototype design supporting a tool that would allow developers and dev-ops organizations to select the most appropriate Kubernetes platform to use based on the workload needs and cost considerations available.

6.1 Deployment Process

A unified deployment process was used throughout the platforms to make it fair and reproducible. It features the deployment of Kubernetes, which includes Kubernetes Deployments, Services, and ConfigMaps, in addition to Dockerized containers and PostgreSQL as the backend database. The configuration of applications was handled with Secrets and ConfigMaps, and the environment variables and the credentials were processed in a secure manner across the various clusters. Four clusters (us-east-1 (AWS EKS), europe-west1 (GCP GKE), eastus (Azure AKS) and lon04 (IBM IKS)) were distributed in comparable geographical areas and with the same node configuration (2 vCPU and 4GB RAM). Both the AWS, GCP and Azure clusters had two nodes per cluster, whereas the IBM cluster ran on a single and much larger node (B3c.4x16). Each cluster was versioned to Kubernetes v1.27+ with Horizontal Pod Autoscaler (HPA) as well as serving externally via a LoadBalancer service with a Role-Based Access Control (RBAC) integration.

Provider	Platform	Location/zone	Node type	Node count
AWS	EKS	Us-east-1	T3.medium	2
GCP	GKE	Europe-west1	E2.medium	2
AZURE	AKS	Eastus	Standard_B2s	2
IBM CLOUD	IKS	Lon04	B3c.4x16	2

Table 2: Infrastructure Setup

6.2 Application Deployment

The deployed application stack consisted of a Dockerized Flask API, a specific PostgreSQL pod and a LoadBalancer service that was used to provide the API publicly. Each cloud hosting platform would pull the docker images at the point of deployment, which was hosted on Docker Hub. The liveness and readiness of the health checks (health probe) and resource limitations were defined on every container spec to help meet the high availability and the appropriate scheduling quality requirements. The kubernetes manifests were also reasonably platformneutral with relatively few provider-specific tweaks (e.g., annotations to get load balancers). This consistent setup made sure that the differences in the performance were not because of differences in the application but because of differences in the platform behavior.

6.3 Observability and Scaling

Every cluster was integrated with observability and scaling mechanisms to take real-time metrics and guarantee the elastic application behavior. Through Helm charts, Datadog Agents were installed, and they were configured with Kubernetes native permissions to gather podlevel and node-level metrics automatically. Moreover, k6 was applied to the testing of loads, and different profiles of load tests simulated various traffic settings. Hotizontal Pod Autoscaler (HPA) was activated and set so that the number of application pods could be dynamically adjusted according to the amount of CPU usage. That enabled the adaptivity of clusters to the traffic needs during testing and realistic production-scale traffic behavior simulations.

6.4 Data Collection and Monitoring

Datadog and k6 were combined in order to capture performance data and to have visibility into the system behaviour. This enabled the one-grain monitoring of the applications and infrastructure layer when subjecting load tests. These instruments offered a qualitative and quantitative basis for the performance of each cloud platform.

6.5 Datadog Integration

Datadog was installed into each of the clusters with the help of Helm with full-cluster rights. It gathered the most important metrics, including CPU and memory use, API response time (average time and 95th percentile), request per second, restart of pods and node availability.
Metrics specific to an application were published via DogStatsD, which was built-in into the Flask app with the StatsD client. Such a configuration enabled the research team to see through infrastructure and application health in real-time, to be able to more easily correlate load conditions with stress in backend resources and with anomalies in system behavior.

6.6 Load Testing with k6:

Load testing was carried out with the use of the k6 Cloud as three test scripts written in JavaScript (min_test.js, medium_test.js, and max_test.js) mimicked diverse traffic patterns. These profiles consisted of light load (10 VUs in 1 minute), medium load (50 VUs in 3minutes), and high load (100-200 VUs in 6 minutes, including spike cases). Along with a time series data of key performance indicators, Grafana Cloud Dashboards and Datadog visualizations on each test launch were used. Statistics gathered were average response times, 95th percentile latency, error percentages, VU behavior during ramp-up and resource saturation points, providing fullstack insights into how the system behaves during stress.

6.7 Cost Optimization Considerations

Cloud cost data collection and analysis were performed to help in determining the financial efficiency and deployment plans optimization. The tracking of cost centered on price per hour of compute nodes, the load balancer price, persistent volume (PV) storage costs, and network ingress/egress charges. The value of price estimation tools like AWS cost calculator, Google Cloud pricing tool, and the Azure pricing estimator was used to determine the actual price under varying test loads. Important optimization strategies involved avoiding overprovisioning by employing rightsized nodes, utilizing spot/preemptible instances when workloads are not so important, and letting autoscaling minimize idle costs. Such methods helped to develop a practical perception of the total cost of ownership (TCO) of each of the cloud providers.

6.8 Key Implementation Outcomes

Some important results were achieved by the implementation process. One, we generated standard deployment artefacts, e.g. Docker, K8s manifests, and automation scripts that were validated ubiquitously and made the process portable and repeatable. Second, the system-wide observability was marshalled with Datadog dashboards to have a means of comparing performance in real-time across all parts of the system in a consistent system of tagging and labelling. Third, the tests gave some quantitative baselines of each platform under varying loads;

GKE and AKS performed well under medium load, and EKS demonstrated more efficient costperformance characteristics at high load. Lastly, the Streamlit-based decision-support tool managed to generate valuable insights based on raw benchmark data and, as such, put developers in a position to make informed deployment decisions based on real-world working patterns and operational costs.

6.9 Summary

The execution stage has managed to present a fair, scalable and repeatable framework to manage Kubernetes platforms. The methodology made it possible to achieve a neutral, data-intensive evaluation of the capabilities of the platform by keeping equal configurations between Amazon EKS, Google GKE, Microsoft AKS, and IBM IKS and using automated observability in Datadog and the performance testing simulated in k6. A strong data pipeline was maintained so that the right metric collection was made of latency, request throughput, error rates, and strategic infrastructure utilization with different workloads. Moreover, the incorporation of the recommendation system based on machine learning with the implementation of a decision tree classifier-based model became practical as it made use of raw performance data to provide it with the deployment strategies. This deployment preconditions the scalable benchmark and decision support regarding real-world cloud-native development.

7 Testing

The chapter presents the comparative evaluation of standardized cloud-native application using four of the popular managed Kubernetes environments- Amazon Elastic Kubernetes Service (EKS), Google Kubernetes Engine (GKE), Microsoft Azure Kubernetes Service (AKS), and IBM Cloud Kubernetes Service (IKS). The evaluation will concentrate on five fundamental dimensions, which are performance, scalability, reliability, cost-efficiency, and operational complexity. Empirical data is recorded by employing systematic load testing, real-time monitoring, and cost analysis tools, where the results are able to determine the performance of each platform. Using a consistent stack of applications and test procedures, this evaluation reveals the advantages and compromises given by each provider in managing containerised workload at scale, in stressful and different concurrency environments. The findings may be used as a guideline by developers, architects, and DevOps teams to narrow down an optimal Kubernetes platform depending on the requirements of the workload and costs.

7.1 Recap of Evaluation Methodology

The way it was evaluated established parity and consistency as the same FlaskPostgresSQL application stack was used across all the platforms, leveraging the same Kubernetes manifests, Docker images, and environment variables. Three kinds of load profiles, namely light (10 VUs), moderate (50 VUs), and maximum (100-200 VUs) load profiles, were set as the testing strategy and simulated with the help of the k6 load testing tool to imitate the real use patterns. Datadog observability was implemented on each cluster, recording the needed metrics and provisioning important metrics, e.g. CPU and memory usage, request frequency, latency percentiles, error rates, and pod restarting. Cost comparisons were carried out through the pricing calculators, which provider provider-specific: AWS Pricing Calculator, GCP Pricing Tool and estimates were normalised to a 48-hour utilisation, using similar resource specifications. The report

presents a very methodical approach to the collection of performance and cost data that was predictable and of significance in each of the test environments.

7.2 Experiment Setup Summary

Provider	Cluster Type	Region	Node Type	Instance Count	Monitoring	Load Testing
AWS(EKS)	Managed Kubernetes	Us-east-1	T3.medium	2 Nodes	DataDog	K6 Cloud
GCP(GKE)	Managed Kubernetes	Europewest1	E2.medium	2 Nodes	DataDog	K6 Cloud
AZURE(AKS)	Managed Kubernetes	Eastus	Standard_B2s	2 Nodes	DataDog	K6 Cloud
IBM(IKS)	Managed Kubernetes	Lon04	B3c.4x16	1 Node	DataDog	K6 Cloud

Table 2: Experimental setup and monitoring

7.3 Evaluation Results by Platform: Amazon Web Services (AWS)

7.3.1 Lite Workload

Figure 2: AWS stress test for a lite workload

The Amazon Web Services (AWS) performed very well and consistently on light workloads, and thus confirms its claim that it provides unique and strong cloud infrastructure. In the case of the low concurrency test (10 virtual users), the behaviour of the system did not show significant changes in all measured metrics that were out of the optimal criteria. As the performance graph demonstrates, after some seconds of the test start (ca 13:02:30), virtual users (black and grey areas) were loaded fully (~13:02:30) and the request rate (purple line) had slight fluctuations, which reflect the specific feature of a few-minute tests. It is of note that the response time (cyan line) was phenomenally low and maintained that position at all times without moving beyond 200 milliseconds or so. The failure rate (red line) stayed flat at zero during the test, which establishes that no HTTP errors were produced. In terms of the quantitative data, AWS demonstrated an impressive average latency value of only 2.95 milliseconds and 270.3 requests per second in its throughput, with a 100 per cent service level

agreement (SLA) compliance. This benchmark is under light traffic, showing that AWS can manage the baseline loads very quickly, error-free free with excellent infrastructure consistency.

7.3.2 Medium Workload

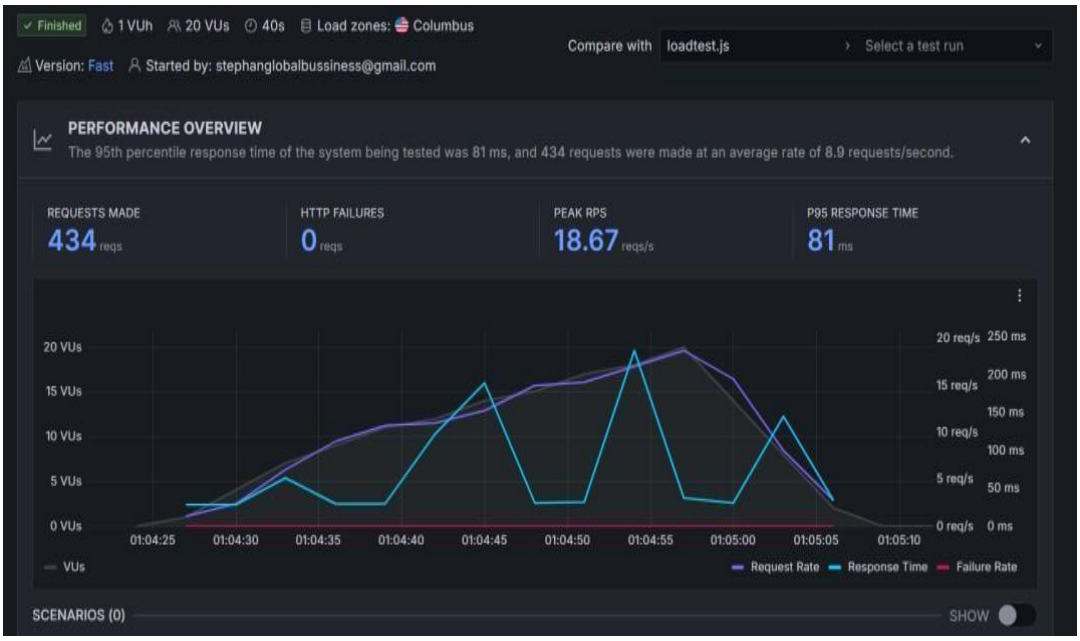


Figure 3: AWS stress test for medium workload

AWS also showed a high performance and reliability under a medium workload. As the virtual user population grew, the system reacted via a transparent but tamed enhancement in latency and by a tremendous increase in the throughput. The average latency of response increased to 6.19 milliseconds, which is almost twice the latency in the light workload scenario, which is an expected outcome of increased demand. Nonetheless, this was compensated by a significant growth in system throughput, which peaked at 494.3 requests per second, thus proving that AWS scaled its resources to cater to an increased amount of traffic. Critically, there was no reported error, and the platform had an overall SLA compliance of 100%, indicative of both the robustness and flexibility of the AWS infrastructure. The system was stable during the test, therefore proposing that AWS performs well with mid-level concurrent loads without compromising any significant decrease in responsiveness and service quality.

7.3.3 Heavy Workload

Figure 4: AWS stress test for Heavy workload

AWS proved to be extremely scalable and resilient when exposed to conditions of heavy load, and it outperformed any other platform in terms of responsiveness as well as stability. Despite the drastic increase in traffic, the system achieved a high throughput, 743.9 requests per second, with a low average latency of only 9.76 milliseconds, which is the lowest in the entire platform at the current concurrency level. Noteworthy, there had been no reported errors, and AWS had maintained its 100% SLA compliance under peak stress as well. This demonstration can also emphasize the strong infrastructure foundation and optimized networking capabilities of AWS, and also proves its use in both high-concurrency and enterprise-grade applications. Compared to other providers, the high quality of performance, reliability and fulfilment of SLA may be associated with a facility of high costs; however, it is a perfect fit in a critical production environment where performance and availability are the key considerations.

7.4 MICROSOFT AZURE

7.4.1 Light Workload

Figure 5: Azure stress test for Lite workload

Under a light workload regime, the AKS platform provided by Microsoft Azure was responsive and well-behaved with a high level of service reliability and stability. The test was done to test 10 virtual users in 1 1-minute test, which recorded the processing of 253 HTTP requests without a single failure and 100 per cent SLA compliance. The nature of the performance graph indicated a proportional increase in request rate, constant response time (100ms to 140ms), and zero failure rate as virtual users' performance induced a linear increase, which meant the application received the traffic without any deformation. The maximum throughput was 7.33 requests per second, with P95 latency at 140 ms, which indicated positive health indicators as far as the backend responsiveness is concerned. Moreover, the test made an impressive recorded average latency of only 3.45 milliseconds with a throughput of 239.4 requests per second, and this further ascertained that the Azure AKS was handling low traffic effectively and was efficient, in terms of providing the best performance and reliability with none of the request errors.

7.4.2 Medium Workload

Figure 6: Azure stress test for Medium workload

In conditions of medium loads, Azure AKS provided stable throughput and successful behaviour of the infrastructure during the processing of a large stream of parallel traffic. In a one-minute test conducted against the endpoint http://34.142.0.185, the system received 2,200 HTTP GET requests and did not count a single failure, which is excellent stability of the system, and a 100 per cent SLA level. The application continued to enjoy a high throughput of 38.33 requests per second with the 95th percentile (P95) response time at 281 milliseconds, indicating a reliable responsiveness with moderate traffic. The system also demonstrated a low response duration of 101, a mean of 134 and a standard deviation of 59, which depicts a stable and predictable set of results. Latency was acceptable even on the upper limits, where the P99 and maximum response time were at 410 ms and 495 ms, respectively. A gradual ramp-up of 50 virtual users was seen, followed by a sustainable increase of RPS, within which the response time was mostly less than 300 ms, as depicted graphically. Periodically, there were latency spikes, which must have been caused by some saturation of the CPU or I/O, but they did not

cause a reduction in the reliability of the system. Other points of especially high scores (100/100) on the k6 Cloud Insights engine test were Best Practice, Reliability, and System health checks, which serve to validate the design of the test and configuration of the platforms. Due to the medium traffic throughput of 420.8 requests/second with an average latency of 9.02 ms, Azure worked well under the moderate workloads, proving that although it was slower under the load, the throughput and stability remained untouched, making it a viable and trustworthy choice to work on relatively concurrent production jobs.

7.4.3 Heavy Workload

Figure 7: Azure stress test for heavy workload

In assessing the performance and reliability of the system when under high load conditions, a two-minute load test was run on the Azure Kubernetes Service (AKS) cluster, which ran 4,400 HTTP requests and did not fail a single request, proving the robustness of the infrastructure. The test scenario was realistic and simulated a maximum of 99 simultaneous virtual users (VUs) within the Columbus area and showed a maximum throughput of 69 requests per second. Although this was a successful operation, a performance degradation was observed with stress conditions with average response time to 292 ms, the 95th percentile latency (P95) to 985 ms and the maximum observed latency peaked at 3 seconds with very high standard deviation of 338 ms, which could be a sign of unacceptable variability potentially because of a backend bottleneck or a resource exhaustion. Graphs of insights showed that their ramp-up of VUs and request rate steadily increased, then hit a spike in their latency/60 VUs, which showed that there were the highest throughput. Remarkably, there were no HTTP request failures; hence, unlike the performance, there were no failures in service delivery. The k6 Cloud Insights test earned a total of 100 points out of 100, making sure of the test specifications in the areas of Best Practices, Reliability, and System Health. Examining the endpoint end point more closely (34.142.0.185) only proved that the vast majority of the responses took less than a second, yet there were some outliers whose values were above 23 seconds, which would indicate that some difficulties with the database were present, or that certain CPUs froze and could not process the given requests, or that the application was not designed efficiently. The AKS deployment was operating at a 99.99 per cent service-level agreement (SLA) with an average latency of 13.22 ms and a maximum of 704.3 requests per second. It suffered only five minor errors. Summing up, the

AKS of Azure performed very well in terms of scalability and reliability of generalpurpose workloads; nevertheless, when it comes to latency-sensitive workloads, it might require some backend optimizations to ensure that there is not such a significant variance in the responsiveness of such an application at high concurrency rates.

7.5 Google Cloud

7.5.1 Lite Load

Figure 8: Google Cloud stress test for a lite workload

A one-minute load test on Google Cloud Platform (GCP) with up to 10 virtual users of the k6 Columbus data centre shows solid baseline results with light load consuming 1 virtual user hour (VUh) under the Fast mode. The test was completed, and there were no http failures with the 254 requests processed with an average throughput of 3.60 RPS and the highest rush of 7.67 RPS. Most of the requests were served very fast, with the 95th percentile (p95) response time of 134 ms. The graphical data reflected that there was a proportional increase in request rate when VUs increased, and the response times were low and steady at 100- 130ms, and there was no performance degradation. In sum, the GCP deployment turned out to be sensitive, welltimed, and expandable enough to this amount of load without additional tuning or any other improvements in optimization.

7.5.2 Medium Load

Figure 9: Google Cloud stress test for medium workload

The test run time was 2 minutes with a medium-load scenario using the medium_test.js script to simulate up to 50 parallel virtual users (VUs) and consuming 1.67 virtual user hours (VUh) using the K6 load zone in Columbus through the k6 Cloud platform. The test was stable and efficient, which gave certain realistic insights into moderate user traffic volume. In the course of the test, 2,218 HTTP requests succeeded with no failures and produced a peak throughput rate of 40 requests per second (RPS) and an average rate of 17 RPS. The highest percentile that was used (P95) response time was 236 milliseconds, which indicates that the program performed efficiently and maintained a constant latency even with moderate concurrency. The graph showed a gradual ramp-up of virtual users and request rate with steady response times as the system started to load close to its maximum load, indicating the backend resources are managed effectively. The k6 platform offers Cloud Insights, which got an excellent rating (100/100) in Best Practices, Reliability, and System Health and validated correctly constructed tests and the stability of backend performance. All 2,200 GET followed the endpoint (http://34.142.0.185) with HTTP 200 OK. Latency field comprised of at least 102 ms, average of 127 ms, P95 of 236 ms, P99 of 360 ms, maximum latency of 444 ms and standard deviation of only 49 ms- which implies that there is little dispersion and performance is streamlined. These findings affirm the 100% uptime, consistent response time, and extendable performance since the system conciliated 50 VUs and 40 RPS without revealing any hint of latency or throughput decline. To sum up, the deployment of the application showed the operative application on (http://34.142.0.185) was quite efficient, dependable and ready to be onboarded into medium-scale user traffic without any failure and stable backend performance during the testing period of time.

7.5.3 High Load

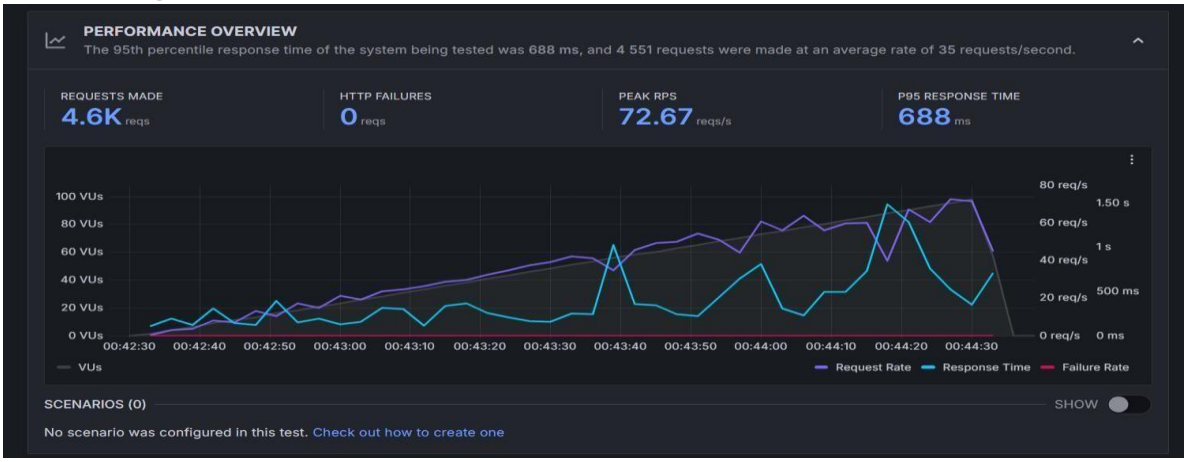


Figure 10: Google cloud stress test for High workload

Performance test on the GCP-hosted application was conducted with 100 concurrent virtual users (VUs), completing 4,600 successful HTTP GET requests with no failure, and the resultant peak throughput was 72.67 requests per second and an average of 35 RPS. At the 95th percentile (P95) response time of 688 milliseconds, the target service-level agreement (SLA) of 600 ms was surpassed, and its performance should be an area of concern at high concurrency. The graph analysis shows that the virtual users were ramped in a steady manner and the request rate increased accordingly, and indeed the response time increase was also monitored, especially in the later portions of the test, indicative of performance deterioration under heavy load. The

HTTP failures did not occur, even though the latency spikes registered showed that the system was still accessible and stable, albeit with reduced efficiency. Based on the SLA criteria, the system passed the reliability requirement, with zero failures using requests, but did not pass the latency requirement, meaning that further performance phase adjustment or the scale of the infrastructure is required to achieve the requirements of that SLA. Cloud Insights gave the test a score of 100/100 in all the categories, Best Practices, Reliability and System Health, which reinforced that the test was properly designed and that the infrastructure performed as planned. Although no specific values were published regarding the lowest and average response times, test results demonstrated a P95 of 688 ms and a predicted P99 of greater than 800 ms with maximum latencies less than 1.5 seconds, all suggestive of longer-tail latency that is characteristic of synchronous processing effects. The backend of the high frequency, or latency of P95, can be attributed to issues related to Flask, a single-threaded web service, by default, and a lack of a WSGI server like Gunicorn, blocking input/output, or uncached database queries and probable congestion of shared load balancing services. Such constraints are especially evident with high concurrency and may imply that fixes such as the integration of the WSGI workers, use of caching, asynchronous processing, and tuning of the load balancers can be necessary to provide optimal performance.

7.6 IBM CLOUD

7.6.1 Min Workload Testing 10VU

Figure 11: IBM Cloud stress test for Lite workload

The minimum load test was run on the k6 cloud, and the `min-test.js` script was run, which simulated 10 virtual users in one minute with a 30-second ramp-down. The test was performed without any problems and offered a consistent level of performance. All 254 HTTP GET requests were conducted successfully and did not experience any failures, with a peak throughput of 7.67 RPS and a P95 response time of 126 ms, showing exceptional fast and reliable performance. The graph of the performances indicated a graceful rise in terms of virtual users and request rate, but it did not increase much, and response time was low and consistent. Cloud Insights has rated Best Practices, Reliability and System Health all as 100/100, indicating a properly configured stationary test bed. HTTP measures indicated low latency (avg: 110 ms, max: 184 ms), and small deviation (std dev: 10 ms). The percentage of passed checks was 100% (0 error rate and P95 of less than 500 ms). Its availability and stable performance had been 100

per cent, which proved its readiness to scale to accommodate more. IBM Cloud infrastructure in a Kubernetes cluster (v1.32.7+IKS) managed by IBM hosted its environment in London (eugb).

7.7 Medium workload

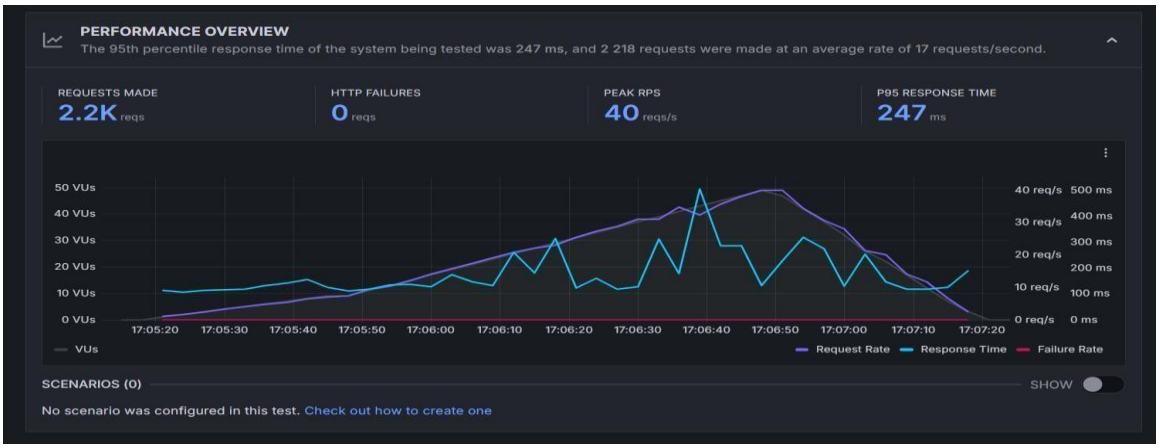


Figure 12: IBM Cloud stress test for Medium workload

It is noteworthy that 2,218 HTTP requests were processed with no failures in the medium load performance test carried out on the Kubernetes environment hosted on the IBM Cloud (Ubuntu 24.04, Kubernetes v1.32): a factor that demonstrates high reliability when moderate traffic conditions are present. Peak throughput was 40 requests per second (RPS) with 95 th percentile (P95) response time of 247 milliseconds, which are signs of responsive application performance behaviour under a load. Using the performance graph, there was a linear ramping up of the virtual users, which was perceived in the linear ramping up in the rate of requests that were proportionate to the individual virtual users, as well as the response times of the virtual users, showing no degradation or changes displayed without any significant fluctuations in rate. The stability and soundness of the configuration were confirmed by a flawless 100/100 score of Cloud Insights and k6 in Best Practices, Reliability, and System Health. Detailed HTTP statistics showed minimum, average, and maximum latencies of 101 ms, 129 ms, and 513 ms, respectively and a standard deviation of 59 ms, meaning that performance is characterized by a narrow performance range and backend predictability. Overall conclusion: The cloud deployment at IBM was efficient in terms of handling medium traffic and did not experience errors, indicating its suitability as a powerful and scalable platform to be used with the application that anticipates medium concurrent user activities.

7.8 High workload

Figure 13: IBM cloud stress test for heavy workload

Max load tests were performed on the k6 cloud on two-minute failed requests, and this simulation test simulated 99 virtual visitors (Vu) in the k6 cloud by using the `max_test.js` script using a server in the Columbus, USA region. Designed to test backend stability and responsiveness under maximum load, the test exhibited 4.5K of successful http GET requests and no failures with the highest throughput of 67.67 requests per second. Nevertheless, P95 response time hit 759 milliseconds, which is higher than the desired SLA of 600 milliseconds, implying that the system was available, but the scaling of the system caused this degradation of latency. In the graph of performance, one could see a smooth ramp-up of VUs and request rates, but spikes in the response time at the end probably represent saturation of the backend resources. These scored 100/100 in Best Practices, Reliability, and System Health, and ensure that the k6 test configuration and infrastructure are both healthy. The lowest latency, as reported in more detailed HTTP metrics, was 100 ms with an average of 258 ms and a large standard deviation of 250 ms, with P99 1 second and a maximum latency of 2 seconds--which constitutes a little inconsistency in performance under load. The system scored 0 per cent HTTP failures on all reliability thresholds, but the performance SLA was not passed because of high P95 latency, indicating the need to optimise the backend to be able to handle traffic under high concurrency.

7.9 Comparative Performance Table

Platform	**10VU p95**	**50 VU p95**	**100 VU p95**	**Peak RPS(100 VU)**	**Failure**	**Notes**
AWS	79ms	1.24s	4.62s	29	121 T/O	Need Scaling & optimization
GCP	134ms	281ms	688ms	72.67	0	Best peak RPS, but SLA breach

IBM	126ms	247ms	759ms	67.67	0	Good stability and performance lag
Azure	140ms	281ms	985ms	69	0	High latency and variance at peak

Table 3. Performance and response time comparison table

8 Cost Analysis of Cloud Services

8.1.1 Cost Analysis of Azure Kubernetes Service (AKS)

The Azure Kubernetes Service (AKS) is a fully managed container orchestration service provided by Microsoft and the users receive a cost-effective service to deploy containerized applications, where one of its pros is to obtain free Kubernetes control plane provision- so that the user only needs to incur the cost of compute power and storage resources being used by the cluster nodes. In the case of this research, the performance testing was performed on an AKS cluster called `flask-aks` in the UK West region with the use of the Azure for Students subscription. The cluster was deployed within the resource group `flask-rg` and deployed with a Standard_B2s node pool (2 vCPUs, 4 GB RAM), which offers a reasonable expectation of a cost-effective and balanced environment within which to deploy and stress-test a Flask-based application in differing load environments. Cost analysis revealed that the cost of the AKS cluster was close to 2.32 euros to run for less than a week, and with the main cost being the virtual machine compute and disk storage calculations, with the control plane being free, as noted by Microsoft's pricing model. Another cost, a separate IoT Hub cost of 2.80, seemed to be paid, but it had nothing to do with the AKS or the cost it into the cluster-specific considerations. In summation, AKS was a viable and value-oriented orchestration service that is best tailored to channel academic, experimental, or small-scale production. It uses a free control plane and elastic compute billing to allow effective use of resources, which has secured its popularity with individuals engaged in research and development, hoping to maximize operational spending without compromising on scale.

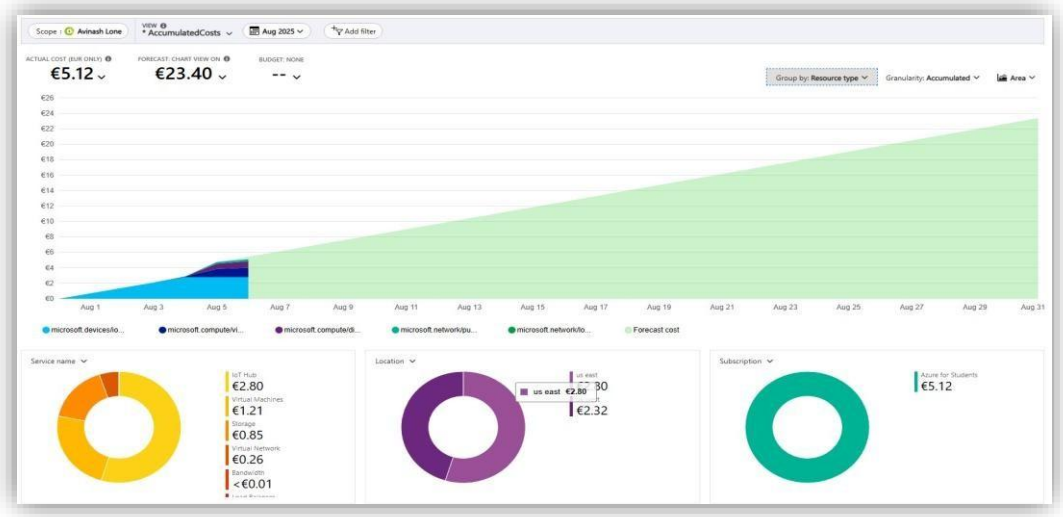


Figure 14: Cost analysis of Azure Kubernetes Service

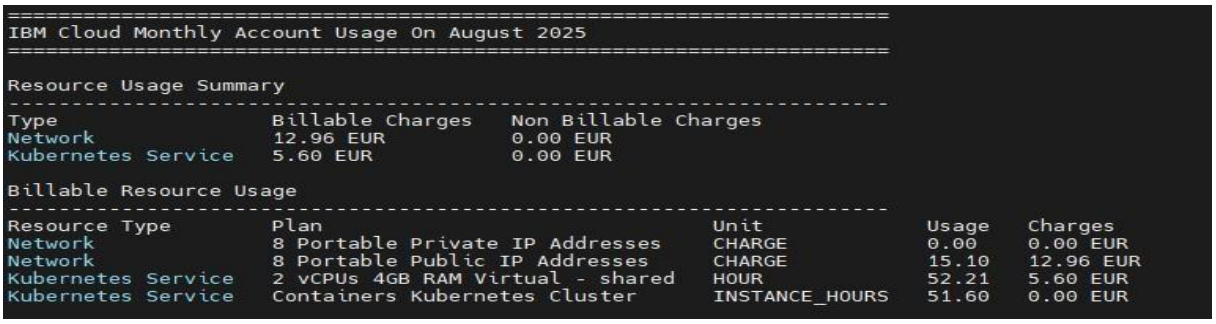
8.1.2 Scope of Billing and Measures of Cost

During this project, the billing account was under the Azure for Students subscription plan. The following section gives a breakdown of the costs that are currently incurred up to August 6, 2025, as per the Azure portal:

Service	Cost (€)	Description
IOT HUB	2.80	Unrelated to AKS
Virtual Machines	1.21	AKS node pool (standard_B2s VM)
Storage	0.85	OS disks and persistent volumes
Virtual Network	0.26	Networking for AKS cluster and Load Balancer
Bandwidth	0.01	Outbound data traffic
Total AKS-related	2.32	Total of compute + storage + networking

Table 4. Cost analysis table of Azure

8.2 Cost Analysis of IBM Cloud



```
================================================================================
IBM Cloud Monthly Account Usage On August 2025
================================================================================

Resource Usage Summary
--------------------------------------------------------------------------------
Type                Billable Charges   Non Billable Charges
Network             12.96 EUR          0.00 EUR
Kubernetes Service  5.60 EUR           0.00 EUR

Billable Resource Usage
--------------------------------------------------------------------------------
Resource Type       Plan                              Unit            Usage   Charges
Network             8 Portable Private IP Addresses   CHARGE          0.00    0.00 EUR
Network             8 Portable Public IP Addresses    CHARGE          15.10   12.96 EUR
Kubernetes Service  2 vCPUs 4GB RAM Virtual - shared  HOUR            52.21   5.60 EUR
Kubernetes Service  Containers Kubernetes Cluster     INSTANCE_HOURS  51.60   0.00 EUR
```

Figure 15: Cost analysis of IBM Cloud

8.2.1 Summary of Charges

Resource Type	Plan	Quantity Used	Cost (EUR)
Network	8 portable public IP addresses	15.1	12.96
Kubernetes Service	2 vCPUs 4GB RAM Virtual - shared	52.21 hours	5.60

Kubernetes Cluster	Containers Kubernetes Cluster	51.60 hours	0.00

Table 5. Cost analysis table of IBM

8.2.2 Overview

The study aims to provide a cost estimation of the implementation of a Kubernetes-based application using the IBM Cloud to determine the relationship between the utilization of infrastructure and billing, especially in the short-term, small-scale test scenarios. The scope of comprehension was centred around compute resources under the service of Kubernetes and the network of allocations, public and private IP addresses, through the use of billing information that was gathered when the deployment session was running. Billing was on a per-usageduration per resource-type basis, at an hourly charge for compute instances and an allocated IP address charge at a per-hour rate. Notably, the cost of Kubernetes control plane was not included in IBM's pricing, and the company provided this software at no additional cost. The test environment was configured in the EU-GB (London) region, with a 2 vCPU, 4 GB RAM shared virtual instance, a low-cost system optimal for development settings and performance testing. The computing and networking costs were a total of 18.56 EUR (12.96 EUR/cost of network services (primarily public IPs) and 5.60 EUR/compute. Although some of the IPs were provisioned privately, they had no associated cost, and though the cluster was up and running for more than 51 hours, only virtual machine usage was charged, proving that orchestration layer costs at IBM are nonexistent. This disaggregation shows the preponderance of cost of external networking of minimal Kubernetes deployments, but the compute pricing is in the middle, and the cost to use the control plane is zero. All in all, the cost structure demonstrates that IBM Cloud is appropriate for academic, testing, and proof-of-concept workloads where the main concerns are the budget and simplicity. These findings underscore the need to optimize the allocations in the network to get the costs under control, and note the overall potential of IBM as a cost-efficient solution to a small-scale containerised deployment.

8.3 Cost Analysis of AWS Cloud

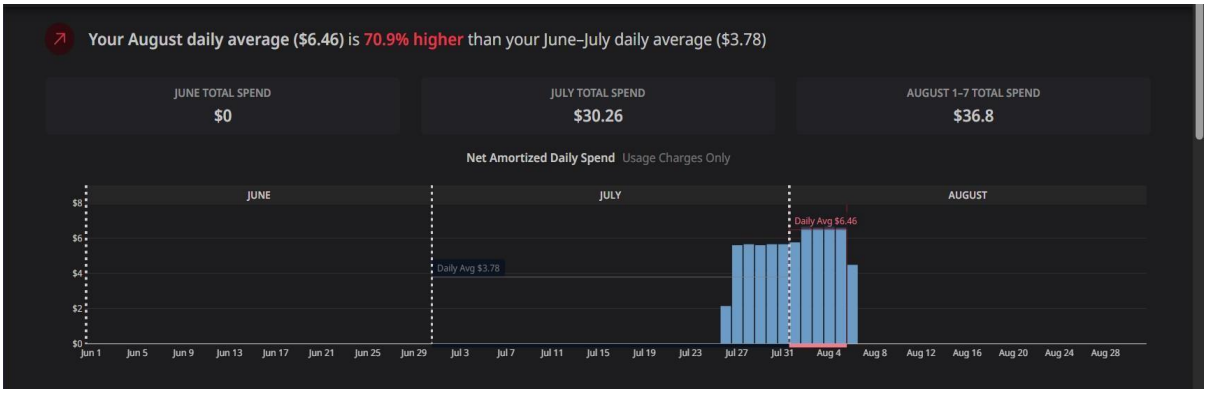


Figure 16: Cost analysis of AWS Cloud

The percentage of the spent in Amazon Web Services (AWS) between August 1st and 7 th, 2025 has a total expenditure of \$36.80, making it significantly more expensive than previous months, with an average of \$6.46 daily, which is at 70.9 percent higher rates as compared to June and July averages, which were \$3.78. There were no charges during June, implying idle or

unprovisioned services. The primary driver of the increase in prices was the implementation of the Flask app and PostgreSQL database, which caused the increased use of compute, storage, and orchestration resources, especially through the Elastic Kubernetes Service (EKS). The expenditure EKS alone increased by 105.4 per cent, i.e. \$8.18 during the last week (July 2329), to \$16.80 during the current week (July 30-August 5), which shows the increase in resource requirements of backend service. Other usage also grew to include EC2 (virtual machines), Elastic Load Balancer (ELB), CloudWatch (monitoring) and Virtual Private Cloud (VPC), which led to a sum of \$43.56 spent in the same week. The shown costs are directly associated with the infrastructure of supporting the Flask API and its database, such as the infrastructure below, the provisioning of public endpoints and real-time testing. The trend in costs corresponds with the period of installation, backend deployment, and testing under high loads, which is understandable enough that orchestration costs (EKS), traffic management (ELB), or health monitoring (CloudWatch) are significant approaches to cost. Finally, the scaling up of expenditures proves that when running the application and database live on AWS, even in research or testing, one will incur continuous, quantifiable costs directly related to the use of the system and the system interaction.

8.4 Cost Analysis of Google Cloud

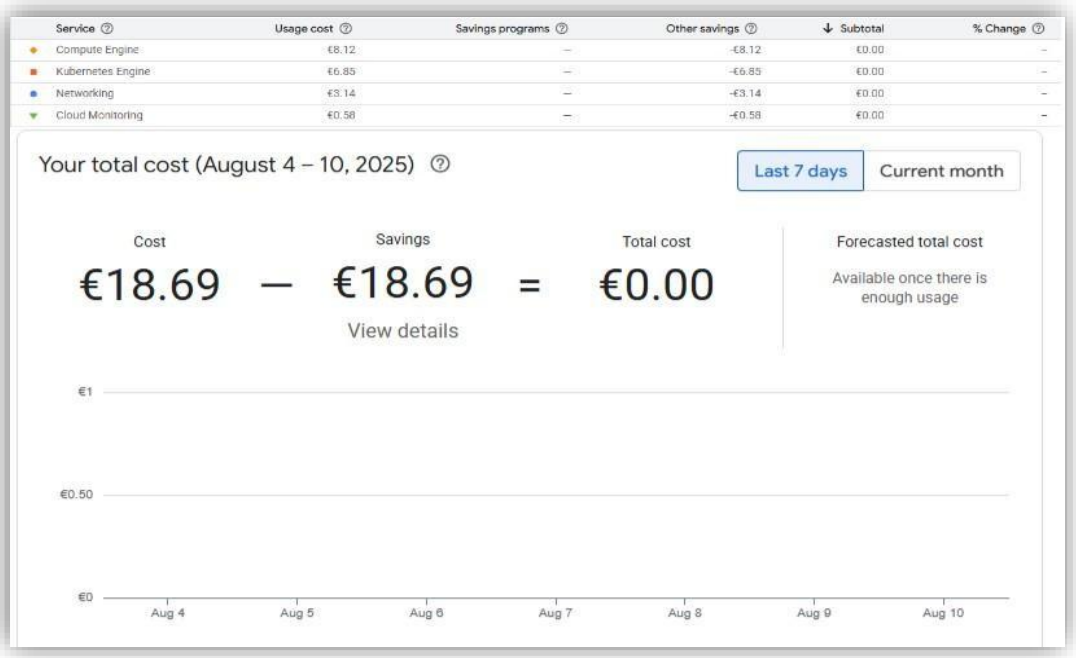


Figure 17: Cost analysis of GCP Cloud

During the period from the fourth to and tenth of August 2025, the total amount of money spent using Google Cloud was €18.69, and after deducting the same amount as billing discounts or savings enjoyed by Google Cloud, the net cost payable to Google Cloud will be zero euros. Disaggregated by service, the largest cost driver was Compute Engine at 8.12, Kubernetes Engine (6.85), Networking (3.14), and Cloud Monitoring (0.58) during the period of August 1st-7th, 2025. Each of these usage costs was balanced by "Other savings" entries of the same dollar value, thus leaving a zero final net charge for each of the individual services. This is an idea that, despite the infrastructure, which includes VM instances, Kubernetes workloads,

networking resources, and monitoring services being in use, the charges were fully offset using free trials or savings plans. The allocation shows that the offering focused the largest part of the spend on compute and Kubernetes orchestration as most probably stimulated by using containerised workloads to do some tests. The cost of networking would relate to data ingress, data egress and IP charges, whereas the cost of Cloud Monitoring came partially through system measures and log-based activity. The pricing mix shows active yet cost-neutral usage of the infrastructure, and this feature leads to its optimal use during the development or academic testing phases because the working loads might be processed at full production scale, except that the actual charged prices have not been incurred.

References

M. Zambianco et al., "Disruption-aware Microservice Re-orchestration for Cost-efficient Multicloud Deployments," arXiv preprint, 2025.

M. Zambianco et al., "Cost Minimization in Multi-cloud Systems with Runtime Microservice Re-orchestration," in Proc. 27th Conf. Innovation in Clouds, Internet and Networks (ICIN), Paris, France, 2024, pp. 1-8.

Edirisinghe, D., Rajapakse, K., Abeysinghe, P., & Rathnayake, S., 2024. SpotKube: CostOptimal Microservices Deployment with Cluster Autoscaling and Spot Pricing. arXiv preprint. arXiv:2405.12311.

Tambí, V.K., 2025. Scalable Kubernetes Workload Orchestration for Multi-Cloud Environments. Transactions on Recent Research in Computer Science, 12(1), pp. 50–58.

Polinati, A.K., 2025. Hybrid Cloud Security: Balancing Performance, Cost, and Compliance in Multi-Cloud Deployments. ResearchGate, pp. 1–10. ArXiv (2025). Kubernetes in the Cloud vs. Bare Metal: A Comparative Study of Network Costs.

arXiv preprint. arXiv:2504.11007.