National College of Ireland

Empirical Study of Cloud Deployment Strategies: Guiding the choice between Containerization, Traditional and Hybrid Deployment

MSc Research Project
Cloud Computing

Arpit Shah
Student ID: 22208224

School of Computing
National College of Ireland

Supervisor: Sean Heeney

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Arpit Shah
Student ID:	22208224
Programme:	Cloud Computing **Year:** 2023-2024
Module:	MSC Research Project
Supervisor:	Sean Heeney
Submission Due Date:	12/08/2024
Project Title:	Empirical Study of Cloud Deployment Strategies: Guiding the choice between Containerization, Traditional and Hybrid Deployment
Word Count:	**9406 Page Count: 23**

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: Arpit Shah

Date: 12/08/2024

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):	

Empirical Study of Cloud Deployment Strategies: Guiding the choice between Containerization, Traditional and Hybrid Deployment

Arpit Shah
Student ID:22208224

Abstract

The empirical study provides a comprehensive evaluation of cloud deployment strategies – containerization, traditional virtual machines (VMs), and hybrid methods for three application types like static web applications, database web application and multithreaded applications with RabbitMQ. Motivated by the need for practical, data-driven guidance for cloud practitioners, the study evaluates key metrics such as performance, scalability, cost, reliability, and operational complexity. The findings shows that containerized deployment offer better performance and scalability for static web applications, hybrid deployments excel in performance, scalability and reliability for database web applications and multithreaded applications but both deployment strategies require complex setups which increases the operational complexity. While traditional VM deployments offer easy setup and low-cost offering usability for smaller applications and applications which do not have much load like academic projects or proof of concepts. A decision tree-based recommendation tool was developed to support practitioners in selecting appropriate deployment strategies based on the empirical data. Despite some of the limitations, including short evaluation period and resource constraints on scalability tests, this study shows a direction for future research in long performance analysis, broader application types, in depth scalability test and enhancing the recommendation tool. This future work will also help in commercializing this research study by the support of recommendation tool. The research ultimately provides actionable insights and practical tools to optimize cloud deployment strategies for its users, to ensure informed decision-making based on application requirements and scenarios.

1 Introduction

Cloud computing has been growing as an element of IT systems that provides flexible, economical and scalable options for a wide range of applications. As businesses and developers transition towards cloud setups, the decision of selecting appropriate deployment strategy is important. The most discussed and used deployment strategies are categorized as containerization, traditional virtual machines (VMs) and hybrid deployment strategy. Each strategy has its unique benefits as well as some challenges, that makes the selection even more complex and often biased based on theoretical discussions.

The motivation behind this research consists of many sides. Firstly, discussing about the larger audience that is the cloud practitioners and developers require a need to make informed decisions regarding the deployment strategies as per their application needs. The new practitioners generally face a lot of challenges due to lack of clear, empirically backed guidance on the advantages and disadvantages of each strategy. For example, containerization utilizes lightweight tools such as Docker and Kubernetes to be efficient in resource utilization and fast in development. However, it can also bring complexity in management and performance consistency (Pahl et al., 2019). Traditional VMs provide reliability and isolation but at a cost of more resource overhead and slower times for deployment (Ahmad *et al*., 2015; Kozhirbayev & Sinnott, 2017).

In addition to this, hybrid deployment strategies, combining an element from containerization and one from the traditional methods, find themselves much more suitable because they pick on the strengths of the approaches. However, there exists limited empirical evidence to guide practitioners on when and how effectively various deployment strategies can be used (*Azumah et al*., 2018). The identified gap is very critical which needs to be addressed particularly for developers or cloud practitioners working on academic projects or small-scale applications that require performance optimization, scalability and cost efficiency without any extra resources (Lohumi *et al*., 2023).

1.1 Research Question

How can empirical performance data across a wide variety of applications be leveraged for guiding the selection between containerized, traditional, and hybrid deployment strategies?

The question stated is important in selecting appropriate deployment strategy for various application scenarios by understanding the strengths, weaknesses and in detail usage of all the three deployment strategies.
It has the following objectives:

1. Perform an in-depth empirical comparison of containerized, traditional, and hybrid deployment strategies using various real-world application use-cases.
2. Analyze the metrics from a perspective of performance, scalability, efficiency, cost, and operational complexity to yield proper actionable insights.
3. Provide practical guidelines and recommendations for cloud practitioners and developers to select the appropriate deployment strategy based on the empirical evidence.

Hypothesizing that containerization will exhibit superior resource efficiency and faster deployment times, traditional VMs will provide better isolation and reliability, and hybrid models will offer a balanced approach, this study aims to test these hypotheses through rigorous empirical evaluation.
There are previous studies that had identified gaps in cloud deployment strategies. Felter *et al*. (2015) compared the performance of VMs and containers, where containers proved to better for significant performance benefits but there was lack in the study which required broader empirical data across several applications scenarios. Pahl *et al*. (2019) reviewed container technologies and stated about the requirement for the in-depth empirical evaluations based on several real-world applications. Also, (Azumah *et al*., 2018) explored hybrid cloud deployments, which identified data location as its major challenge and propose policy-based methods to improve deployment efficiency.
To gain an understanding further, this study uses a combination of mixed methods integrating quantitative performance measures with qualitative insights. This study involves deploying several types of applications like dynamic web applications, microservice architecture, blockchain applications, multithreaded applications across different cloud deployment strategies to derive empirical data. Metrics such as scalability, latency, cost and resource utilization will be analysed to offer a detailed comparison. It aims to establish new benchmarks for cloud deployment strategies and enhance the understanding of all 3 strategies and their proper usage, which will lead to optimized deployment solutions.
The report is structured as follows:

1. Introduction: It outlines the research purpose, motivation, question, objectives, hypothesis and contributions.
2. Related Work: This section states the study within existing literature and critically reviews similar work.
3. Methodology: This section details the research design, procedures and techniques used.
4. Design Specification: This section mentions the several techniques and frameworks used
5. Implementation: This section mentions the entire deployment process in detail.

6. Evaluation: It will show the detailed analysis of the results presenting key findings in the form of metrics.
7. Conclusion and Future work: It restates the research question and objective by discussing the effectiveness and limitations of the study as well as suggest directions for future research.

2 Related Work

This shows the present study withing the existing literature, providing critical review of related work on containerized, traditional and hybrid deployment strategies in cloud computing. The aim is to highlight the strengths and weaknesses of each study along with the gaps that this research aims to address and how this study will fulfil it.

2.1 Containerization vs Traditional Virtualization

Felter *et al*. (2015) conducted a study where performance comparison was done between virtual machines (VMs) and Linux containers using KVM as the hypervisor and Docker as the container manager. They found containers offered better performance mainly with respect to startup time and resource efficiency. The performance benefits of containers were context-dependent which suggest a detailed empirical study to generalize these results across wide variety of applications. This study provides a comprehensive performance analysis, but it also highlights the need for more empirical evidence. This study addresses the gap by providing empirical data across different applications to generalize the findings.
Kozhirbayev and Sinnott (2017) extended this comparison by evaluating container-based technologies for cloud environments. The study highlights the flexibility and lightweight nature of containers, that makes them more suitable for resource managements and dynamic scaling. But these benefits were specific for scenarios and mainly based on flexibility which stated the requirement for further evaluation across different context based on several applications. This will be address by evaluating container performance in various types of applications.
Narasimhulu *et al*. (2023) investigated the impact of containerization on the deployment process in DevOps. Their study stated that while containers streamline the deployment pipeline, but they also introduce complexities in orchestration and monitoring, which requires robust tools and practices to manage effectively. Their study mainly focused on practical insights into DevOps integration, but it highlights the challenge of orchestration. Our research will analyse the efficiency of different orchestration tools mainly Kubernetes to address this gap.

2.2 Performance Evaluation of Deployment Strategies

Shah *et al*. (2021) provided comprehensive benchmarking and performance evaluation of various VMs and containers for cloud-based scientific workloads The consequence of that is, while in general containers are a better solution with performance in mind, VMs give better isolation and stability, which is important for some scientific applications. Their study was strong in detailed benchmarking but performed for scientific applications only; this study will fill this gap by providing detailed benchmarking for a wide range of applications.
Performance study for comparison between the two architectures of microservices and serverless was carried out by Fan *et al*. (2020). Results showed that microservices deployed within containers have more resources and are optimized. Thus, such microservices provide better performance than their serverless counterparts, despite cold-start problems. As related to the results derived from their study, performance optimization is the main focus in the below-referenced study; however, it brought to light latency challenges on serverless architecture. This

research would analyze the latency for diverse architectures that encompasses containerized and hybrid models, hence offer a more in-depth evaluation.

Al Qausar *et al*. (2023) investigated performance metrics for containerized applications and revealed detailed evaluation of metrics, such as scaling and efficiency in containerized environments. Our results show that containerized deployment is performing satisfactorily across various metrics, but at the same time, it also points out the issues with performance stability and resource management.

2.3 Hybrid Deployment Models

Vu *et al*. (2022) introduced a predictive hybrid autoscaling method for containerized applications, which combined vertical and horizontal scaling techniques using machine learning to predict future demand and optimize resource utilization. The results were enhancements in maintaining response time below QoS constraints while achieving high resource utilization. The strength of this study is its advance predictive scaling, but its implementation is complex. We will explore simpler hybrid scaling techniques to address this complexity.

2.4 Comprehensive Review and Trends

Pahl *et al*. (2019) provided a comprehensive review of cloud computing technologies that addresses various aspects such as architecture, orchestration and performance optimization. They stated the need for empirical studies to evaluate the practical implications of container technologies in different application scenarios. This study's strength is its thorough review, but it also calls for more empirical data which will be provided by our study after practically evaluating data in real-world scenarios to fill this gap.

Watada *et al*. (2019) observed emerging trends, techniques, and open issues in containerization, identifying key areas for improvement and future research. They also discussed regarding the need for better performance optimization and orchestration tools. This study's strength is its future opportunities, but it also highlights significant open issues. The challenges will be addressed as part of our study.

2.5 Comparative Studies of Deployment Models

Patel and Kansara (2021) conducted a comparative study of cloud deployment models that provided insights into the strengths and weaknesses of different approaches. They concluded that as containerization offers flexibility and resource efficiency, traditional VMs offer better isolation and stability. This further indicates that the comparative approach is the main strength of this study, which also shows the need of hybrid models to balance such trade-offs. To Meet these tradeoffs of flexibility and isolation, the use of hybrid deployment strategy is required to overcome the challenges in their study.

2.6 Summary and Gaps

The reviewed literature shows significant advancements in containerization, traditional and hybrid deployment models. While containers provide several benefits in performance and resource efficiency, they also create management and handling challenge of stateful application instances. Traditional VMs perform better in terms of isolation and stability but it is done at the cost of resource overhead. A hybrid deployment strategy is needed for maintaining a balance between the constraints under both strategies that shall give more empirical data for guidance.

This study tries to fill the gaps by providing a detailed empirical comparison between containerized, traditional, and hybrid deployment strategies based on different types of real-world applications. The practical guidance for cloud practitioners and developers are provided through the evaluation of metrics which includes performance, scalability, cost, reliability and operational complexity over a wide range of applications.

Study	Focus	Key Findings	Strengths	Weaknesses	Gaps	How We Are Addressing
Felter *et al.* (2015)	VMs vs. Containers	Containers offer better performance in startup time and resource efficiency	Comprehensive performance analysis	Context-dependent results	Need for broader empirical data	Providing empirical data across diverse applications
Kozhirbayev and Sinnott (2017)	Container-based Technologies	Containers suitable for dynamic scaling; specific scenario benefits	Flexibility and lightweight	Scenario-specific benefits	Need for evaluation across contexts	Evaluating container performance in various application scenarios
Narasimhulu *et al.* (2023)	DevOps and Containers	Streamlined deployment but complex orchestration	Integration with DevOps	Orchestration challenges	Need for robust orchestration tools	Analyzing orchestration tool efficiency
Shah *et al.* (2021)	VM and Container Configurations	Containers offer better performance, VMs provide isolation	Benchmarking scientific workloads	Limited to scientific applications	Broader application scenarios	Evaluating various application types
Fan *et al.* (2020)	Microservices vs. Serverless	Microservices offer better control over resource allocation	Performance optimization	Cold start latency in serverless	Need for latency optimization	Assessing latency in different architectures
Al Qausar *et al.* (2023)	Container-based Applications	Containers deliver satisfactory performance for various metrics	Extensive evaluation of metrics	Performance stability and resource management issues	Need for better evaluation of containerized environments	Investigating performance stability and resource management challenges through experiments
Ebert *et al.* (2016)	DevOps	Importance of CI/CD with containers	Supports consistent environments	Requires robust orchestration	Need for better orchestration tools	Assessing advanced orchestration solutions
Pahl *et al.* (2019)	Container Technologies	Comprehensive review of container technologies	In-depth analysis of architectures	Need for empirical studies	Practical evaluation in real-world scenarios	Providing empirical evaluation data
Watada *et al.* (2019)	Containerization Trends	Identified emerging trends and open issues	Better perspective	Significant open issues	performance optimization	Addressing performance optimization challenges
Patel & Kansara (2021)	Deployment Models	Comparative study of deployment models	Comparative approach	Trade-offs between flexibility and isolation	Need for hybrid models	Exploring hybrid deployment strategies

Table 1: Summarization of related works

3 Research Methodology

Building on the foundation set in the related work section, the research wants to give an empirical evaluation of all the three cloud deployment strategies: traditional (EC2), containerized (Docker + MicroK8s), and hybrid based on the type of different applications. By deploying different kinds of applications, it is assumed that the assessment will be comprehensive with respect to performance, scalability, reliability, cost, and operational complexity of each strategy. The methodology includes a detailed research approach, equipment and tools used, experimental setup, data collection, data evaluation and interpretation along with the recommendation tool to ensure a in depth investigation.

3.1 Research Approach

The primary objective of this study is to empirically evaluate the effectiveness of three cloud deployment strategies by using three various types of application such as static web application, database web application and multithreaded application with RabbitMQ to build a better guiding framework for the cloud practitioners and developers to allow them select appropriate strategies based on the application and not only follow the theoretical concepts but also have proper real world observations. The applications were selected to provide a wide range of real-world scenarios that allows in depth assessment of each strategies strengths and weaknesses. The study involves deploying each application using traditional, containerized, and hybrid methods on Amazon Web Services (AWS), followed by proper monitoring, testing and evaluation.

3.2 Equipment and Tools

The wide ranges of tools were used to ensure a proper experimental setup. The cloud infrastructure was deployed on Amazon Web Services (AWS) because of its flexibility and universal adoption. The t3.large instance was used for traditional deployments and t3.xlarge instance was used for containerized and hybrid deployments. Docker was used for containerizing applications and MicroK8s was used to manage the Kubernetes environments for containerized deployment [1]. Prometheus and Grafana was used for monitoring and visualization of various metrics [4] [5]. The load testing was conducted by using Apache JMeter to stimulate real-world user traffic [2]. Horizontal Pod Autoscaler (HPA) was used for managing the horizontal scaling in containerized environment [3] and the development of recommendation tool was done by using python and decision tree algorithm which also included Streamlit for creating user-friendly web interface for the tool.

3.3 Experimental Setup

The experimental setup involves configuring three unique deployment environments on AWS EC2. The traditional deployments are directly done on AWS EC2 t3.large instance where Prometheus, Grafana, JMeter are configured for monitoring and easy of deployment of the application. The containerized deployments uses Docker to package the applications into the containers and it is managed with MicroK8s on t3.xlarge instance. This approach provides isolation and scalable environments which ensures consistency across the deployments. The hybrid deployment combines the traditional EC2 instance and containerized environments which used t3.xlarge instance for containerized components as well as same ec2 instance for traditional components. The aim of this setup is to gain the benefits of both the methods.

3.4 Data Collection

The data collection was a crucial part of this research which uses Prometheus to collect the real-time data such as CPU usage, memory consumption, and network I/O which is then visualize on Grafana dashboard for a detailed evaluation and collection of the data [4] [5], The scraping of data is done using various exporters such as node exporter, mongodb exporter, RabbitMQ exporter and the in-built observability namespace of the Kubernetes. Apache JMeter was used to generate load and simulate user requests to measure key performance metrics including latency, throughput and error rates under various load conditions.

3.5 Data Evaluation and Interpretation

The metrics collected was systematically evaluated to assess the effectiveness of each deployment strategy. Performance metrics includes latency and throughput where latency is used

[1] **https://microk8s.io/docs**
[2] **https://jmeter.apache.org/**

to measure the time taken to process a request and provide a response. Throughput is used to assess the number of requests processed per unit time. Scalability was examined through horizontal scaling capabilities and load test parameters to evaluate the ability to add more instances or containers within the existing instances. Reliability was focused more on system uptime, error rates and fault tolerance that provides insights into the robustness and stability of each strategy. Cost analysis include both deployment and operational cost which is measured using AWS calculator and it covers the initial setup cost along with the resource usage and maintenance cost. Operational complexity was measured by evaluating the ease of deployment and maintenance requirements. A proper descriptive statistics and comparative study was conducted to summarize data and compare performance metrics across different deployment strategies. The techniques ensured that the evaluation and interpretation was thorough, and the conclusions drawn were based on concrete evidence.

3.6 Development of the Prototype Recommendation Tool

In addition to evaluating the deployment strategies, to support the study a prototype was developed for recommendation tool. This tool was designed to guide in selecting the optimal deployment strategy based on the application requirements. The recommendation tool was built using a decision tree algorithm, which was trained on the metrics collected from each deployment for various applications. The development process involved several steps like, preparing of dataset from the metrics collected for all types of applications and deployment strategies. Categorical values were converted into numerical values suitable for decision tree algorithm. The decision tree model was trained to learn the relationship between these metrics and recommended deployment strategy. Once model is trained, it was saved and use for recommendation system. The system has a web interface developed using Streamlit which allows users to input the characteristics of their applications. The tool uses decision tree model and will recommend the most suitable deployment strategy.

By following this research methodology, the aim of the study is to provide clear and detailed evaluation of all three deployment strategies. This methodology sets a stage for the Design Specification section, which build on this foundation by detailing all the technical configurations and setup procedures that are used to implement the deployment strategies described in the methodology.

4 Design Specification

Following the research methodology, the design specification explains more on the infrastructure setup, deployment process, metrics collected, and tools used for evaluating and comparing the three cloud deployment strategies: traditional (EC2), containerized (Docker+MicroK8s), and Hybrid (EC2+Docker+MicroK8s). It will ensure a detailed understanding of the technical configurations, tools used, and the process involved in the study. The figure 1 shows the overview of the components of the entire study which highlights the application types, along with its platform services used and the infrastructure setups.

[3] **https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/**
[4] **https://grafana.com/docs/**
[5] **https://prometheus.io/docs/introduction/overview/**

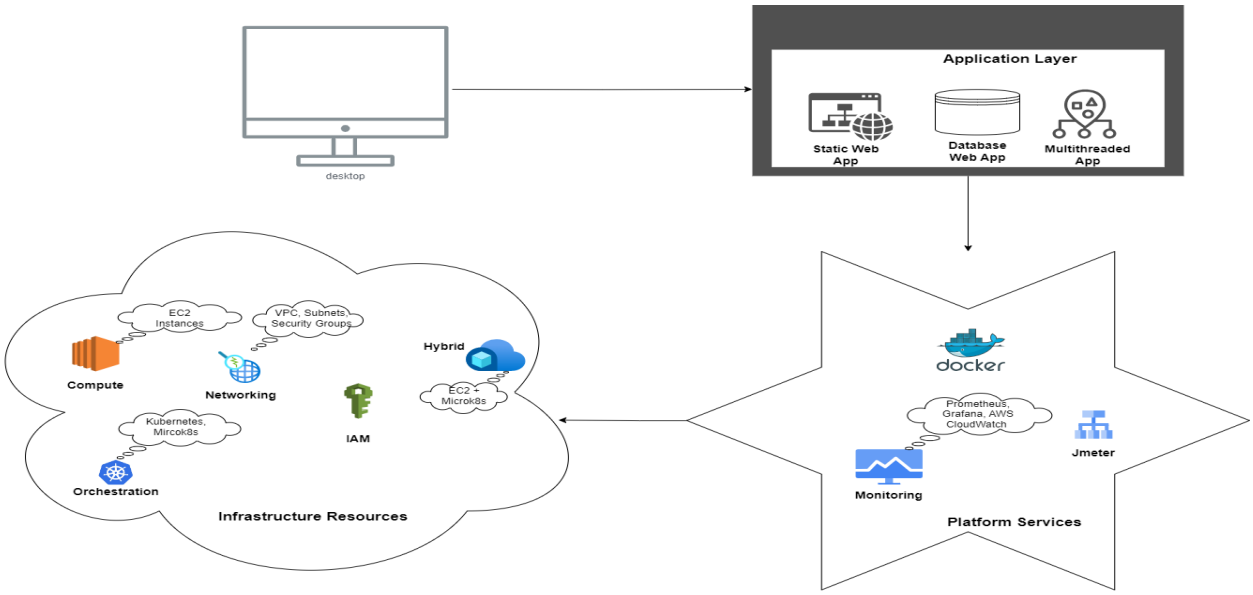

Figure 1: Overview of Cloud Deployment Strategies

4.1 Infrastructure Setup

The infrastructure setup for this study includes three different deployment strategies on Amazon Web Services (AWS) which is designed to replicate a real-world cloud environment to evaluate various metrics like performance, scalability, reliability, cost and operational complexity.

In the traditional deployment strategy, Amazon EC2 is used to host applications in a traditional virtual machine setup. For this setup, t3.large instance was selected to make sure performance, cost and other metrics are properly evaluated. All application were directly deployed and configured on EC2 instance which involved setup of required environments, installation of required software packages as well as optimizing system configuration for performance. This includes the capabilities of the EC2 instance to handle workloads in more efficient way.

For containerized deployment strategy, the combination of Docker and MicroK8s were utilized to package and manage the applications withing the container. This setup used t3.xlarge instance to provide the necessary resource for containerization. Docker provides the isolated environments for each application to manage them more conveniently which ensures consistency and portability. MircoK8s, a lightweight Kubernetes distribution is used as it is a single node cluster to manage these containers that enables smooth deployment, scaling and management [1]. This approach provided with better efficient resources utilization for this study in terms of cost and restrictions along with simplifies deployment process by encapsulating the application with its dependencies inside the containers.

The hybrid deployment strategy is the combination of traditional EC2 instance and containerized environment to utilize the strengths of both the approaches. In this setup, t3.xlarge instance was used for containerized components and the same instance was used for traditional components. This aim of this approach was to optimize performance and scalability by utilizing the best aspects of both the strategies. Containerized components were managed with Docker and MicroK8s to provide flexibility and scalability, while traditional components benefitted from the stability and straightforward configuration of EC2 instance and taking some load off the containers.

4.2 Applications Deployed

The correct and detailed evaluation of the deployment strategies is very important and hence three types of applications selected and deployed were static web application, a database web application and a multithreaded application with RabbitMQ.

The static web application was selected as a basic test case for evaluating the metrics for a simple basic web application under different loads. The application consisted of simple HTML, CSS, JavaScript and Bootstrap files that were hosted on web server which allowed for straightforward performance measurements using various tools which are explained in later sub sections.

The database web application includes a web interface where todo can be added and save along with its due date. It has a backend database implemented using MongoDB. The backend is build using NodeJS and expressJS framework. The frontend is build using ReactJS majorly. The language used for scripting and programming is JavaScript. This application assessed database operations, response times, data handling capabilities in various deployment environments. This combination of web and database gave the insights into the performance of data-intensive tasks.

The multithreaded application with RabbitMQ was developed to test messaging and concurrency handling capabilities by simulating a stock trading scenario. RabbitMQ acted as the message broker for managing communication between all components. The multithreaded setup provided simulating simultaneous processing of multiple threads, which provided insights into the reliability and performance of each deployment strategy under concurrent loads.

4.3 Metrics Collected

The study is focused on collecting the metrics to evaluate all aspects of each deployment strategy for doing a comparative analysis in form of a recommendation tool. These metrics include performance, scalability, reliability, cost and operational complexity.

Performance metrics like latency and throughput were measured to get detailed understanding of the efficiency of each deployment strategy. Latency measured the time taken to process a request and provide a response, while throughput mentions the number of requests processed per unit time.

Scalability metrics evaluated the ability of each deployment strategy to handle the increasing loads by either horizontally scaling by adding more instance or containers or scaling it vertically by increasing the resources of existing instances. In this study, the horizontal scaling was used in containerized and hybrid strategy by using HPA, while no scaling was given in traditional strategy due to resources constraints. Vertical scaling was not provided as it required to increase a greater number of resources.

Reliability metrics monitored the availability and fault tolerance of the applications. System uptime majorly measured the availability of the application along with its components, error rates tracked the frequency of errors occurring during the operation and fault tolerance was used to measure the system's ability to continue operating even if the component fails.

Cost metrics includes both operational as well as deployment cost. Deployment cost measured the initial phase cost of setting up the infrastructure whereas the operational cost showed the ongoing cost of running applications that included resource usage and maintenance.

Operational complexity evaluated the ease of deployment, maintenance requirements, and deployment speed. These metrics provides information about the efforts required to manage and maintain the application across different deployment strategies.

4.4 Monitoring and Load Testing tools

The monitoring of each deployment strategy for all applications is one of the important factors of this study. Prometheus and Grafana were used for monitoring real-time performance and visualizing them. Prometheus collects the metrics from the deployed applications by scraping and then store them in a time series database. These metrics includes CPU usage, memory consumption, network I/O that provides detailed insights into resource utilization and

performance. To scrape more metrics, the node exporter, MongoDB exporter and RabbitMQ exporters were used which scraped more in detailed metrics for the usage of those database and queue. Grafana dashboards were configured to visualize the metrics which offered an interactive interface to analyse the performance of data and identify trends or anomalies in the applications behaviour.
Apache JMeter was used to simulate user requests and generate load on the applications. It allowed for the creation of test plans that's simulates like a real-world usage scenario which enables measuring latency, throughput and error rates under various load conditions [2]. Also, the Horizontal Pod Autoscaler (HPA) was configured for the containerized applications which automatically scaled the number of pods based on CPU utilization. HPA ensures that applications handle varying loads efficiently by dynamically adjusting resources [3].

4.5 Prototype Recommendation Tool

The major goal of this study to guide cloud practitioners and developers with the correct strategies based on various applications brings us to a prototype recommendation tool that was developed using decision tree algorithm. This tool leverages the collected metrics data in form of csv file format as shown in table 2 to provide deployment recommendations, guiding everyone in selecting the most suitable deployment strategy based on application requirements.

Application	**Performance**	**Scalability**	**Cost**	**Reliability**	**Operational Complexity**	**Recommended Deployment**
[Application Type]	[High/Medium/Low]	[High/Medium/Low]	[High/Medium/Low]	[High/Medium/Low]	[High/Medium/Low]	[Deployment Strategy]

Table 2: Deployment Strategy Evaluation Dataset Template

The development started by preparing a dataset that included the metrics fetched from each experiment mentioned in section 6. These categorical values were encoded into numerical format that is suited for decision tree algorithm. The model was trained to learn the relationships between metrics and the recommended deployment strategies. Once training is done, the model was saved to be used for user interaction.
The recommendation tool features a user interface built with Streamlit where users can select from the dropdown the application type and its characteristics for each metric category. The tool processes this data and uses the trained decision tree model to recommend and display the most suitable deployment strategy.
By detailing the infrastructure setup, application deployment, metrics collection, monitoring tools utilization and developing recommendation tool, this design specification set a comprehensive framework and to understand the experimental approach mentioned in the section 3 of this study. The next section, Implementation, will provide a more practical steps and configurations used to deploy the applications across all three strategies. This will include specific procedures, technical variation and methodologies used for this setup and achieve the objectives outlined in this design specification.

5 Implementation

The implementation of the proposed methodology is mainly focused on the final stages of deploying three types of applications – static web application, database web application and multithreaded application with RabbitMQ in three cloud deployment strategies that is traditional

(EC2), containerized (Docker + MicroK8s), and Hybrid (EC2 + Docker + MicroK8s). The implementation section describes the implementation of all three strategies – the software and configurations used along with the outcome produced whose primary focus is on deployment process and getting metrics for each application in each strategy.

5.1 Deployment Process

The deployment process for each application was properly executed across three cloud strategies to make sure accurate and reliable assessment of all metrics.

5.1.1 Traditional Deployment

The traditional deployment is configured and set up on EC2 t3.large instance of AWS. For static web application, Nginx was used as web server and was installed on EC2 instance. The database web application developed using Node.js with Express.js and React.js involved installing all the dependencies of these modules using npm install with further installing MongoDB on EC2 instance as a database for this application. The multithreaded application developed in Python was deployed on EC2 instance after installing required libraries like Python, Pip and Pika. The installation of RabbitMQ was also directly done on EC2 along with enabling the RabbitMQ management and Prometheus metrics for it. Prometheus and Grafana were also set up and installed directly on EC2 along with various exporters such as MongoDB exporter, Node exporter and RabbitMQ exporter to gather metrics in a very specific manner.

5.1.2 Containerized Deployment

The containerized deployment involved setting up t3.xlarge EC2 instance and keeping docker configured for generating images and pushing them to Docker Hub. The microK8s was setup directly on EC2 instance. For static web applications, the entire application was containerized using Docker by build the image and pushing it to Docker Hub which was then deployed using MicroK8s by using yaml files and applying them. Prometheus was set up on an EC2 instance for monitoring and Grafana which was included with Kubernetes under the observability namespace was used for visualization. The database web application was fully containerized by creating three docker images for the frontend (React.js), backend (Node.js with Express.js) and MongoDB. These images were further deployed using MicroK8s by configuring multiple yaml files and applying them for deployment. Prometheus and MongoDB exporter were set up on EC2 for monitoring and same Grafana was used which is included with Kubernetes under observability namespace for visualization. The multithreaded application was also containerized with docker images for the application and RabbitMQ. These images were again deployed using MicroK8s in similar way and Prometheus and RabbitMQ exporter were set up on EC2 instance and Grafana which was included with Kubernetes under observability namespace was used for monitoring.

To simplify the conversion of Docker Compose file to Kubernetes file configurations, Kompose was used. This tool allows for a streamlined conversion from Docker environment to Kubernetes managed deployments which ensures consistency and reduces manual configuration efforts [6]. Additionally, Horizontal Pod AutoScaler (HPA) was implemented by configuring hpa.yaml file to enable horizontal scaling of containerized applications based on CPU usage to make sure that the applications could handle varying loads effectively.

5.1.3 Hybrid Deployment

The hybrid deployment is a combination of traditional and containerized deployment strategies which includes t3.xlarge instance with Docker and MicroK8s for containerizing. For the database web application, MongoDB was installed directly on EC2 instance and its public IP was given in

the backend code for accessing it, while the remaining components that is the entire code was containerized. Two docker images were created for the frontend and backend respectively, which were further deployed using MicroK8s on the same EC2 instance and same process was followed as mentioned in 5.1.2 containerized deployment section for containerizing. Monitoring was set up with Prometheus and MongoDB exporter on EC2 instance and Grafana was used which was included with Kubernetes observability namespace. For multithreaded applications, RabbitMQ was installed on EC2 with its RabbitMQ exporter and mentioning it accessible IP in the code that was containerized and deployed using MicroK8s on the same instance. Monitoring included Prometheus, RabbitMQ exporter and Grafana from the Kubernetes observability namespace. Kompose was again used to convert Docker Compose files to Kubernetes configuration to simplify the Kubernetes configuration and reduce manual work. HPA was configured to enable horizontal scaling of containerized components, to ensure they could dynamically adjust changes in the load. The static web application was not deployed using the hybrid strategy as it was unnecessary.

5.2 Monitoring and Data Collection

Monitoring and Data collection are critical aspect of this study for evaluating all the metrics of the deployment strategies. Prometheus was configured to gather detailed metrics such as CPU usage, memory consumption, and network I/O from each application for each deployment strategy. These metrics are stored in a time-series database by Prometheus and then visualized using Grafana dashboard. Apache JMeter was used to simulate user traffic and generate load on the applications like real-world scenarios. A test plan (test-plan.jmx) was created for each load testing to capture key performance metrics including latency, throughput and error rates by increasing the threads and loops to test under various load conditions.

5.2.1 Prometheus Configuration

Prometheus was installed on EC2 directly and then configured using Prometheus.yml file which was crucial for setting up Prometheus and accessing it scraping metrics on its web interface across all deployment strategies. It defined various scrape configurations for different exporters like node exporter, MongoDB exporter and RabbitMQ exporter and application endpoints to ensure collection of comprehensive metrics. This file was used consistently to configure Prometheus to scrape metrics from various sources, which provided a stable monitoring solution.

5.2.2 Exporter

The node exporter was installed directly on EC2 instance to expose hardware and OS metrics (such as CPU, memory disk and network usage) to Prometheus. The MongoDB exporter was used to monitor MongoDB performance, providing metrics like query performance, operation details (insert, update, delete), memory usage and connection statistics. RabbitMQ exporter provides metrics such as message rates, queue sizes and connection details to Prometheus for monitoring. These all exporters even provide one common metrics which is uptime that is very useful in showing reliability metrics. They played an important role in providing the data to evaluate the system performance under different deployment strategies. It ensured detailed and relevant metrics were continuously collected and available for evaluation and interpretation [7] [8] [9].

5.3 Development of the Prototype Recommendation Tool

To assist and guide in selecting appropriate deployment strategy, a prototype recommendation tool was developed using a decision tree algorithm. The development process began by collecting and transforming all metrics into a dataset. This dataset included metrics for performance, scalability, cost, reliability, and operational complexity based on different application types and deployment strategies. The decision tree model was trained using this dataset and then saved for further use. The recommendation tool was built using python and DecisionTreeClassifier library.

[6] **https://kompose.io/**

An interactive web interface was developed using Streamlit, which allows users to input specific application characteristics and recommend appropriate deployment strategies to them. This is done by processing the input data and applying the decision tree model against it that provides the suggested deployment strategy based on empirical data collected (Pedregosa et al., 2011).

5.4 Outcomes

The implementation provides several key outcomes which focuses primarily on the deployment and evaluation of the applications:

5.4.1 Transformed Data

Real-time performance metrics were collected from Prometheus and load testing results from Apache JMeter. This data was divided into five metrics that is performance, scalability, reliability, cost and operational complexity which was then properly structured into a dataset that shows various aspects of the deployment strategies such as latency, throughput, CPU, memory usage, uptime and error rates. The metrics and dataset provide detailed comparison of different strategies based on application types.

5.4.2 Deployment Configurations

Detailed deployment configurations were developed for each application and deployment strategy which is mentioned in config manual. This includes Dockerfiles for containerizing applications, Kubernetes manifests for orchestrating containerized deployments, and scripts for setting up traditional EC2 deployment. These configurations ensure re-usability and consistency for different deployment environments. Kompose was used to convert Docker Compose files to Kubernetes configurations which streamlined the process and ensured consistent deployment setups.

5.4.3 Monitoring Setup

A robust monitoring setup is established using Prometheus and Grafana. The prometheus.yml configuration file was crucial in defining the scrape configurations and integrating different exporters for collecting various metrics. Grafana dashboards were customized by adding dashboard ids for various types to visualize different metrics by providing an intuitive interface for monitoring application health and overall performance.

5.4.4 Key Performance Metrics

The study focused on key performance metrics such as performance, scalability, efficiency, cost and operational complexity. These all metrics were fetched from the monitoring tools that were set up for scraping all metrics and visualizing them on Grafana. Scalability was assessed through horizontal scaling using HPA and conducting load test using JMeter, efficiency through resource utilization metrics from Prometheus, cost through AWS cost calculator and operational complexity through deployment and maintenance procedures.

5.4.5 Model Created

A decision tree model was developed and trained based on the collected metrics from each deployment for each application. The model was designed to recommend optimal deployment strategy based on specific application requirements considering factors such as performance, scalability, cost, reliability and operational complexity.

[7] **https://github.com/percona/mongodb_exporter**
[8] **https://github.com/prometheus/node_exporter**
[9] **https://github.com/kbudde/rabbitmq_exporter**

5.4.6 Recommendation Tool

An interactive web application was built using Streamlit that lets user input application details and receive optimal deployment strategy based on the metrics collected. The tool uses the trained decision tree model to provide proper recommendations by guiding users to make informed decisions based on empirical data.

5.4.7 Comprehensive Documentation

Detailed documentation was provided that includes deployment process, configuration, monitoring setups and usage of recommendation tool in the form of config manual. This documentation serves as an important resource for replicating and extending the study and applying findings to similar projects.

The implementation section highlighted the practical steps taken to deploy and evaluate the application for different cloud strategies, that ensures the research study objectives are met. The next section, that is Evaluation, will present the experiments conducted and analyze the results obtained by comparing them to provide a thorough assessment of the deployment strategies.

6 Evaluation

The evaluation section delves into the detailed comparative analysis of all the experimental results, that provides valuable insights into their importance and suggestions. Building on the previous implementation section, the evaluation of the effectiveness of different deployment strategies for all three types of applications are shown. The comparison of metrics like performance, scalability, cost, reliability, and operational complexity are represented using the tables and graph to represent them in more clear and concise manner.

6.1 Evaluation of Static Web Application

This states the evaluation of the static web application for two different environments such as traditional EC2 deployment and containerized deployment. The hybrid deployment was not experimented for static web application because of its inherent nature which won't benefit that much from a hybrid approach. Static web applications typically do not require the dynamic scaling and flexibility of offered by hybrid deployments and hence traditional and containerized comparisons are more relevant and insightful.

6.1.1 Experiment 1: Traditional Deployment

In traditional deployment for static web application, the application was deployed directly on AWS t3.large EC2 instance and the evaluation is focused mainly on metrics like performance, scalability, cost, reliability, and operational complexity.

Category	Value	Rating
Performance	116 ms (average response time) 42.9 ops/s (throughput)	Low
Scalability	Limited scalability due to no scaling provided	Low
Cost	$4.01 (2 days)	Low
Reliability	54 mins uptime	Medium
Operational Complexity	Overall, it is low	Low

Table 3: Evaluation result of traditional deployment

This deployment recorded an average time of 116 ms and a throughput of 42.9 operations per second which indicates good performance for moderate load conditions. There was no scalability used in this experiment due to resource limitations and hence it relied on single instance(t3.large). The load test parameters included 5000 loops, 2000 threads and 120-second ramp time which created heavy load on system and due to no scaling load was increase substantially. The estimated cost for running the traditional deployment on a t3.large instance for two days was around $4.01 which makes it cost-effective for low to moderate usage scenarios. Also, the system maintained an uptime of 54 mins during the entire usage and deployment phase which shows medium reliability even during the load testing. Operational complexity was low with minimum CPU and memory resources commitment along with very easy setup in initial phase which makes it easy to manage but it lacks in scalability and efficiency.

6.1.2 Experiment 2: Containerized Deployment

The containerized deployment involves deploying static web application using Docker and Kubernetes (MicroK8s) on an AWS t3.xlarge instance. Prometheus and Grafana were used for monitoring, and JMeter was used for conducting load testing.

Category	Value	Rating
Performance	26 ms (average response time) 1.15 ops/s throughput	High
Scalability	Horizontal scaling (1 to 10 pods under load)	High
Cost	$47.94 (2 days)	High
Reliability	100% uptime	High
Operational Complexity	Overall complexity was medium	Medium

Table 4: Evaluation Result of containerized deployment

The containerized deployment shows high performance with an average response time of 26 ms and a throughput of 1.15 operations per second. This indicates a good improvement in responsiveness when we compare to traditional deployment. This environment scaled from 1 to 10 pods under load as specified by load test parameters (5000 loops, 2000 threads, 120s ramp time) which demonstrates the horizontal scaling and show high scalability in this setup. The estimated cost for running the containerized deployment on a t3.xlarge instance for two days was around $47.94 which higher than traditional as instance is also different and hence it cannot be considered as cost effective given the performance and scalability metrics. It maintains 100% uptime during the test period which ensures high reliability and continuous availability. The operational complexity was medium considering the CPU requests and memory resource usage along with its initial setup complexity which makes it more complex when compared to traditional deployment.

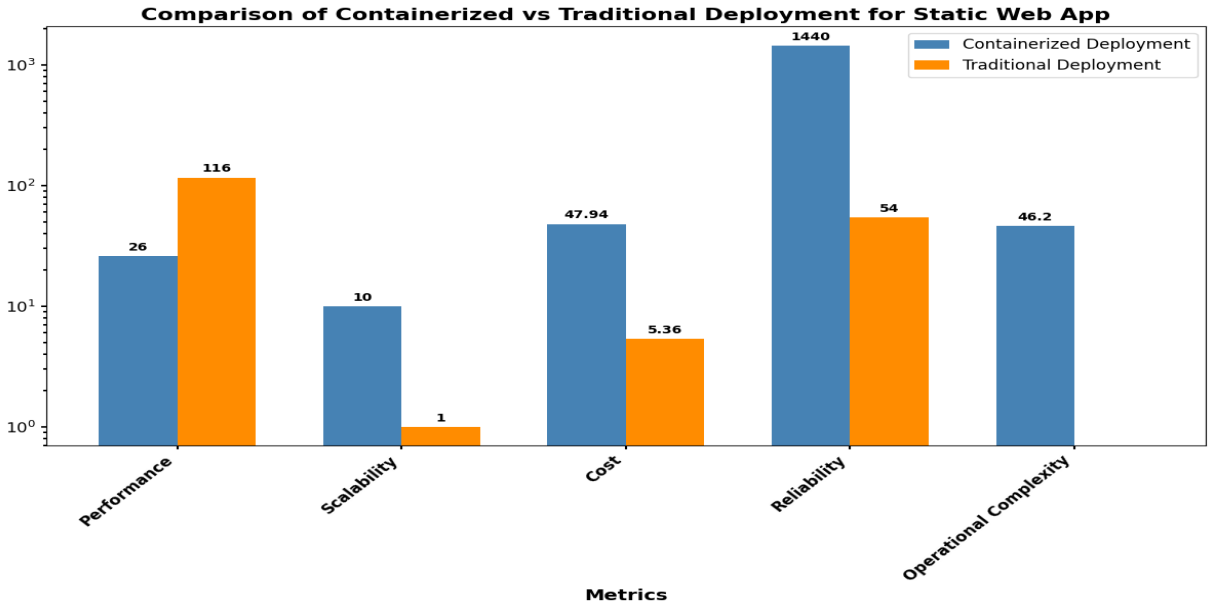

Figure 2: Traditional vs Containerized Deployment comparison for Static web application

The graph illustrates the comparison between traditional and containerized deployments for static web application across various metrics. The containerized deployment strategy for static web application demonstrates better performance, scalability and reliability when compared with traditional deployment. The traditional deployment remains cost effective and better in operational complexity, but it is limited in terms of performance, scalability and reliability. The hybrid deployment is not performed as it is static web application and there nothing to divide and split the deployment into traditional and containerized hence the comparison is done only between traditional and containerized. This helps user to get guidance based on the applications requirements and not only based on one or two parameters.

6.2 Evaluation of Database Web Application

This evaluates the deployment strategies for the database web application across three different environments like traditional EC2 deployment, containerized deployment, and hybrid deployment. The evaluation was focused on performance, scalability, cost, reliability, and operational complexity for each deployment strategy.

6.2.1 Experiment 1: Traditional Deployment

The traditional deployment for database web application was deployed on AWS t3.large instance and it was focused on the metrics such as performance, scalability, cost, reliability, and operational complexity.

Category	Value	Rating
Performance	165.3 ms (average response time) 167 requests throughput	Low
Scalability	Limited scalability due to no scaling provided	Low
Cost	$5.36 (2 days)	Low
Reliability	85% uptime (150 minutes)	Medium
Operational Complexity	Overall complexity was Medium	Low

Table 5: Evaluation Result of traditional deployment

The traditional deployment showed an average response time of 165.3 ms with a maximum throughput 167 requests per second indicating medium performance. Scalability was limited as there was no horizontal and vertical scaling available. The estimates cost for running the traditional deployment on an AWS t3.large instance was $5.36 which shows low pricing. The system maintained 85% uptime i.e., 150 minutes during the test period which shows high reliability. Operational Complexity was low because of the CPU utilization and the ease of initial setup.

6.2.2 Experiment 2: Containerized Deployment

The containerized deployment involves deploying the database web application using Docker and Kubernetes (MicroK8s) on AWS t3.xlarge instance. Monitoring was conducted using Kubernetes tools, Prometheus and Grafana with load testing done using JMeter.

Category	Value	Rating
Performance	9.89 ms (average response time) 4.16 requests throughput	Medium
Scalability	Horizontal scaling (1 to 10 pods under load)	High
Cost	$47.94 (2 days)	High
Reliability	100% uptime (180 minutes)	High
Operational Complexity	Overall complexity was Medium	Medium

Table 6: Evaluation Result of Containerized Deployment

The containerized deployment showed high performance with a work queue latency of approximately 9.89 ms and 4.16 request throughput. Scalability was high with horizontal pod autoscaling (HPA) which was configured between 2 and 10 pods based on load. The estimated cost for running the containerized deployment on a t3.xlarge instance for two days was $47.94 which is considered high. Reliability was high with all pods running continuously during the test period. Operational complexity was medium because of it slightly complex setup along with its manageable CPU and memory resource commitments.

6.2.3 Experiment 3: Hybrid Deployment

The hybrid deployment is a combination of traditional EC2 with containerized approach which uses EC2 and container orchestration with Kubernetes. The database web application was evaluated for performance, scalability, cost, reliability, and operational complexity.

Category	Value	Rating
Performance	7 ms (average response time) 1000 requests throughput	High
Scalability	Horizontal scaling (1 to 10 pods under load)	High
Cost	$47.94 (2 days)	High
Reliability	100% uptime (180 minutes)	High
Operational Complexity	Overall complexity was High	High

Table 7: Evaluation Result of Hybrid Deployment

The hybrid deployment demonstrated high performance with an average response time of 7 ms and a high throughput of 1000 request per second. Scalability was high, with HPA managing between 2 and 10 pods as per the increase in load. The cost for running the hybrid deployment was $47.94 for two days, which is considered high. Reliability was also high as the system's uptime was 100% during the entire load test, but the operational complexity was high due to significant CPU usage and the overall setup steps.

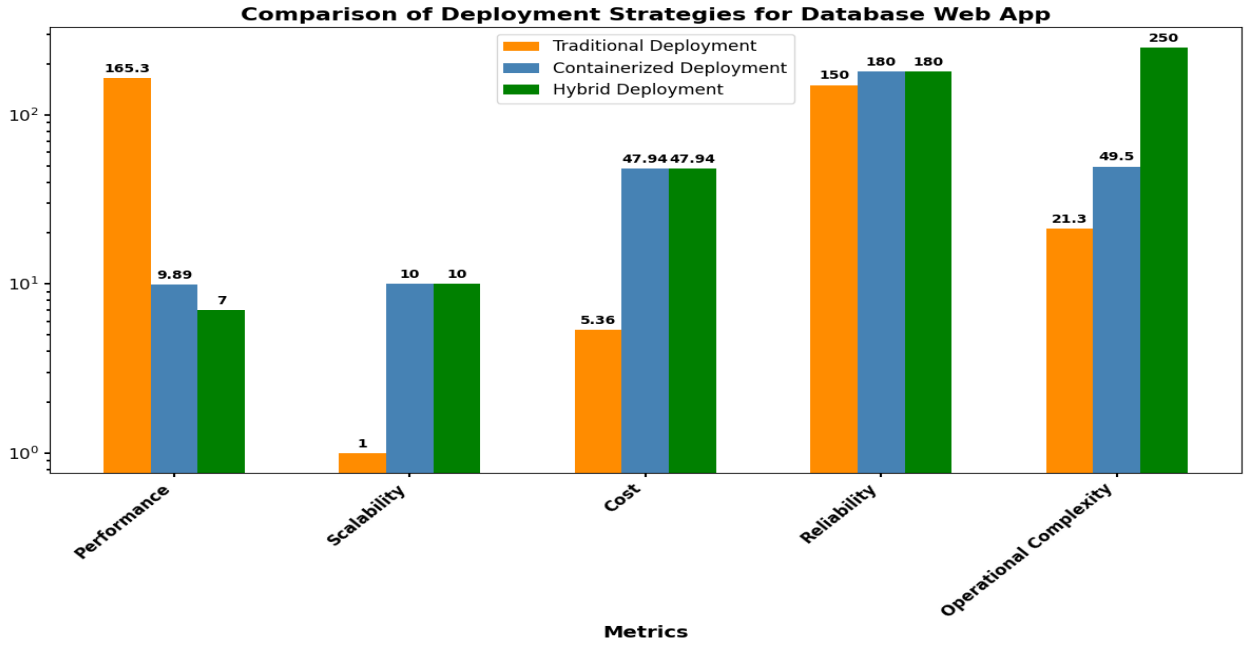

Figure 3: Traditional vs Containerized vs Hybrid Deployment comparison for Database web application

The Hybrid deployment strategy for the database web application provides maximum advantages for performance, scalability and reliability compared to traditional and containerized deployment approach. It is the most expensive option but the benefit in other metrics justifies the cost as well. The containerized deployment also performs well, particularly in cost and operational complexity when compared with hybrid approach and performance, scalability and reliability when compared to traditional approach. The traditional approach remains cost effective and require operational complexity and hence it is suggested to use whenever there are small database web applications or academic projects where scalability is not required mainly.

6.3 Evaluation of Multithreaded Application with RabbitMQ

This evaluates the deployment strategies for the multithreaded application with RabbitMQ across three different environments such as traditional EC2 deployment, containerized deployment, and hybrid deployment. It focuses on metrics like performance, scalability, cost, reliability, and operational complexity for each deployment strategy.

6.3.1 Experiment 1: Traditional Deployment

In this experiment, the multithreaded application with RabbitMQ was deployed on an AWS t3.large instance. The table below shows the key metrics for traditional deployment

Category	Value	Rating
Performance	Approx. 300 ms (average response time) 1.6 messages/s throughput	Low
Scalability	Limited scalability due to no scaling provided	Low
Cost	$3.50 (2 days)	Low
Reliability	80% uptime	Medium
Operational Complexity	Overall complexity was Low	Low

Table 8: Evaluation Result of Traditional Deployment

The traditional deployment shows a response time of approximately 300 ms and a throughput of 1.6 messages per second, indicating low performance. Scalability was not used due to resource constraints. The estimated cost for running the traditional deployment on an AWS t3.large instance for two days was $3.50 making it a low-cost option. The system maintained 80% uptime during the test period, that demonstrated medium reliability. Operational complexity was medium due to the setup steps and moderate CPU utilization.

6.3.2 Experiment 2: Containerized Deployment

The containerized deployment involved deploying the multithreaded application with RabbitMQ using Docker and Kubernetes (MicroK8s) on an AWS t3.xlarge instance and monitoring was conducted using Kubernetes tools, Prometheus, and Grafana, with load testing using JMeter.

Category	Value	Rating
Performance	81.3 ms (average response time) 2.08k messages/s throughput	Medium
Scalability	Horizontal scaling (2 to 10 pods under load)	High
Cost	$47.94 (2 days)	High
Reliability	100% uptime	High
Operational Complexity	Overall complexity was Medium	Medium

Table 9: Evaluation Result of Containerized Deployment

It shows medium performance with a response time of 81.3 ms and a high throughput of 2.08k messages per second. Scalability was high due to it horizontal scaling while load testing with horizontal pod autoscaler (HPA) configured to manage between 2 and 10 pods based on load. The estimated cost for running the containerized deployments on a t3.xlarge instance for two days was $47.94 which is considered to be high. Reliability was high due to the 100% uptime will all pods running continuously during the test period. Operational complexity was medium due to slightly complicate setup then compared to traditional deployment.

6.3.3 Experiment 3: Hybrid Deployment

The hybrid deployment combines traditional and containerized approaches by utilizing both EC2 instances and container orchestration with Kubernetes. The multithreaded application with RabbitMQ was evaluated for performance, scalability, cost, reliability, and operational complexity.

Category	Value	Rating
Performance	75 ms (average response time) 2.08k messages/s throughput	High
Scalability	Horizontal scaling (2 to 10 pods under load)	High
Cost	$47.94 (2 days)	High
Reliability	100% uptime	High
Operational Complexity	Overall complexity was High	High

Table 10: Evaluation Result of Hybrid Deployment

The hybrid deployment shows high performance with a response time of 75 ms and a high throughput of 2.08k messages per second. Scalability was high because of usage of horizontal pod autoscaler (HPA) which manages between 2 and 10 pods based on the load test done by JMeter. The cost for running the hybrid deployment was $47.94 for two days which is high. Reliability is high because of its 100% uptime will all pods running continuously during the test period. Operational complexity was high due to complicated initial setup.

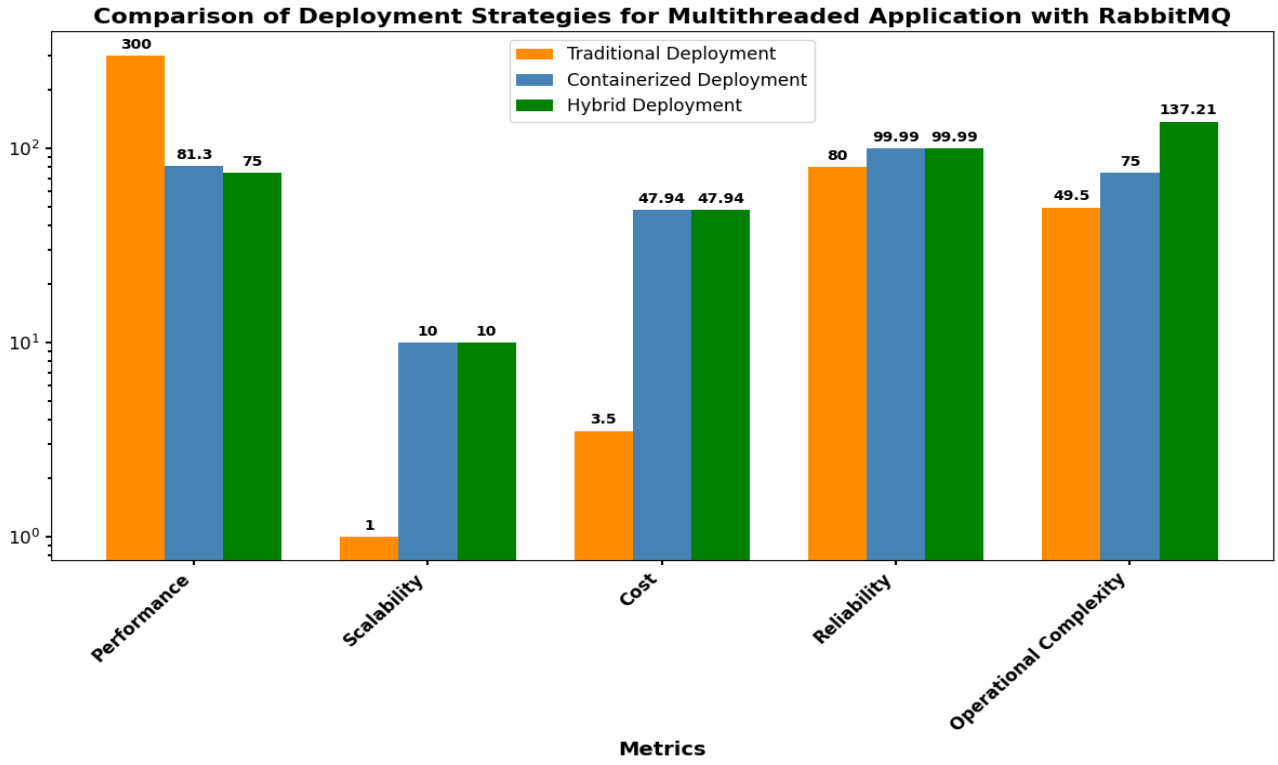

Figure 4: Traditional vs Containerized vs Hybrid Deployment comparison for Multithreaded application with RabbitMQ

The hybrid deployment strategy for the multithreaded application with RabbitMQ offers better advantages in terms of performance majorly when compared with traditional and containerized applications. It even maintains high reliability and low cost with a manageable level of operational complexity. The containerized deployment also performs well, specially in throughput and has medium operational complexity. The traditional strategy remains cost effective option but is also limited in scalability and performance, which make it suitable for less demanding scenarios.

6.4 Evaluation of Recommendation Tool

To complement the research and support the target audience like cloud practitioners and developers in selecting the appropriate deployment strategy, a prototype recommendation tool was developed. This tool makes use of empirical data from all the above experiments to provide proper recommendations based on specific application requirements. The tool's user interface is built using Stremlit, and the model is trained using decision tree algorithm based on metrices such as performance, scalability, cost, reliability, and operational complexity. This tool improves the decision-making process by making sure it not only suggests deployment strategy only on one or two parameters and considers all parameters and then suggest avoiding the standards set for each application type theoretically. By leveraging concrete data, the tools address the gap between theoretical knowledge and practical application which helps user to optimize their deployment strategies for various application scenarios.

6.5 Discussion

The findings from the experiments provides a comprehensive insight into the performance, scalability, cost, reliability, and operational complexity for traditional, containerized, and hybrid deployment strategies. As showed in section 6.1, the containerized deployment for static web application demonstrates superior performance and scalability compared to traditional

deployment, but at a high cost and operational complexity. The rationale behind not including the hybrid approach for the static web application is that a simple solution does not benefit from a hybrid setup. As such, the hybrid deployment of database web applications in section 6.2 clearly outperforms in terms of performance and scalability; however, it is expensive and complex, due to the trade-off between the benefits of performance and overhead management. As a balanced approach, containerized deployments are more expensive but less heavy to maintain compared to the traditional setup. The findings are aligned with other related work in that they clearly show significant advantages of containerization and hybrid methods in dynamic and resource-intensive scenarios.

Section 6.3 showed slightly better performance and scalability for a hybrid deployment of the multithreaded application with RabbitMQ at an increased operational complexity and cost. The traditional deployment model is more cost-effective and easy to setup than performant and scalable. A containerized deployment is more balanced but requires higher accuracy with the resource management process. The recommendation tool, which is explained in section 6.4, is based on the data, which provides a deployment strategy to the user as it recommends based on different application scenarios and requirement. Overall, these experiments show the need for optimization in containerized and hybrid environments to reduce complexity and costs. Future research needs to explore broader range of applications with longer experiments durations, and advanced monitoring to improve the robustness of the findings which provides more actionable recommendations for cloud deployment strategies.

7 Conclusion and Future Work

This study empirically evaluated traditional (EC2), containerized (Docker + MicroK8s), and hybrid deployment strategies for various real-world applications. The primary objectives were to conduct a detailed comparison, analyse key metrics and provide practical guidance for cloud practitioners and developers. The findings show that the containerized deployment excel in performance and scalability for static web application, and hybrid deployments perform better in terms of scalability, performance and reliability for database web application and multithreaded applications with RabbitMQ, whereas traditional deployments show moderate performance with no scalability but always better in terms of operational complexity. The developed recommendation tool guides users in selecting appropriate deployment strategies based on the empirical data. The study not only identifies which deployment strategies are better for each application but also aims at providing some actionable guidance based on application requirements and scenarios. Unlike previous studies where the strategies comparison was shown which is superior based on various parameters, the goal achieved was to empower the cloud practitioner with detailed, scenarios-specific recommendations based on empirical evidence.

The study still has some limitations where evaluation period was relatively short and based on limited number of applications. Also, extensive scalability tests were not performed especially vertical scalability due to resource constraints.

To address this limitations and future improvements, future work should be focused on various key areas:

1. Long-term Performance and Cost Analysis: Conduct more extensive research to evaluate long performance by keeping the system up and test running for number of days to gain more better performance, reliability and cost implication as done in real-world scenarios for all the three deployment strategies.
2. Broader Application Types: The research needs to be extended to include various more types of applications like AI/ML workloads, Blockchain, big data processing, stateless and stateful applications to provide more actionable recommendation across all scenarios.
3. Vertical Scalability: Explore vertical scalability options by increasing the resources for existing instances based on the maximum load created during the testing.

4. Interactive Recommendation Tool: Improve the recommendation tool to allow user to deploy their proof of concepts (POC) applications and test them to gain more personalize insights into the suggestions based on their real-time performance data. Train the model will all types of application to commercialize the tool in future which can be beneficial for practitioners, professional as well as experts.

These efforts will help improve and optimize cloud deployment strategies to provide deeper insights and more practical and actionable guidance for cloud practitioners and developers to make sure they make informed decisions based on their application requirements and scenarios.

References

Ahmad, R.W., Gani, A., Ab. Hamid, S.H., Shiraz, M., Yousafzai, A. and Xia, F., 2015. A survey on virtual machine migration and server consolidation frameworks for cloud data centers. *Journal of Network and Computer Applications*, 52, pp.11-25. DOI: 10.1016/j.jnca.2015.02.002.

Al Qausar, M.J., Soeparno, H., Gaol, F.L. and Arifin, Y., 2023. Software Metrics for Container-Based Applications: Systematic Literature Review. In *2023 International Conference on Information Management and Technology (ICIMTech)*, Malang, Indonesia, pp. 125-130. DOI: 10.1109/ICIMTech59029.2023.10277948.

Azumah, K.K., Sørensen, L.T. and Tadayoni, R., 2018. Hybrid Cloud Service Selection Strategies: A Qualitative Meta-Analysis. *2018 IEEE 7th International Conference on Adaptive Science & Technology (ICAST)*, Accra, Ghana, pp.1-8. DOI: 10.1109/ICASTECH.2018.8506887.

Ebert, C., Gallardo, G., Hernantes, J. and Serrano, N., 2016. 'DevOps', in *IEEE Software*. pp. 94-100. DOI: 10.1109/MS.2016.68.

Fan, C.-F., Jindal, A. and Gerndt, M., 2020. 'Microservices vs serverless: A performance comparison on a cloud-native web application', in *Proceedings of the 10th International Conference on Cloud Computing and Services Science (CLOSERI)*. Prague, Czech Republic, 7-9 May 2020, pp. 204-215. DOI: 10.5220/0009792702040215.

Felter, W., Ferreira, A., Rajamony, R. and Rubio, J., 2015. 'An updated performance comparison of virtual machines and Linux containers', in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. Philadelphia, PA, USA, 29 31 March 2015, pp. 171-172. DOI: 10.1109/ISPASS.2015.7095802.

Kozhirbayev, Z. and Sinnott, R., 2017. A performance comparison of container-based technologies for the Cloud. *Future Generation Computer Systems*, 68, pp.175-182. DOI: 10.1016/j.future.2016.08.025.

Lohumi, Y., Srivastava, P., Gangodkar, D. and Tripathi, V., 2023. Recent Trends, Issues and Challenges in Container and VM Migration. *2023 International Conference on Computer Science and Emerging Technologies (CSET)*, pp.1-5. DOI: 10.1109/CSET58993.2023.10346895.

Narasimhulu, M., Mounika, D.V., Varshini, P., A.K., and Rao, T.R.K., 2023. 'Investigating the impact of containerization on the deployment process in DevOps', in *2023 2nd International Conference on Edge Computing and Applications (ICECAA)*. Namakkal, India, 19-21 July, pp. 679-685. DOI: 10.1109/ICECAA58104.2023.10212240.

Pahl, C., Brogi, A., Soldani, J. and Jamshidi, P., 2019. 'Cloud container technologies: a state of-the-art review', in *IEEE Transactions on Cloud Computing*. pp. 677-692. DOI: 10.1109/TCC.2017.2702586.

Patel, H.B. and Kansara, N., 2021. 'Cloud computing deployment models: A comparative study', *International Journal of Innovative Research in Computer Science & Technology (IJIRCST)*, 9(2), pp. 45-50. DOI: 10.21276/ijircst.2021.9.2.8.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Duchesnay, E., 2011. 'Scikit-learn: Machine learning in Python', *Journal of Machine Learning Research*, 12, pp. 2825-2830.

Shah, S., Waqas, A., Kim, M., Kim, T.-H., Yoon, H. and Noh, S.-Y., 2021. 'Benchmarking and Performance Evaluations on Various Configurations of Virtual Machine and Containers for Cloud-Based Scientific Workloads', *Applied Sciences*, 11, p. 993. DOI: 10.3390/app11030993.

Silva, V., Kirikova, M. and Alksnis, G., 2018. Containers for Virtualization: An Overview. *Applied Computer Systems*, 23, pp.21-27. DOI: 10.2478/acss-2018-0003.

Vu, D.-D., Tran, M.-N. and Kim, Y., 2022. Predictive Hybrid Autoscaling for Containerized Applications. *IEEE Access*, 10, pp.109768-109778. DOI: 10.1109/ACCESS.2022.3214985.

Watada, J., Roy, A., Kadikar, R., Pham, H., and Xu, B., 2019. 'Emerging trends, techniques and open issues of containerization: A review', *IEEE Access*, 7, pp. 152443-152472. DOI: 10.1109/ACCESS.2019.2945930.