

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Automated FinOps for Cloud Infrastructure Cost Management

Guilherme de Matos Ferreira de Almeida



Mestrado em Engenharia Informática e Computação

Supervisor: André Monteiro de Oliveira Restivo

Co-Supervisor: João Pedro Dias

July 17, 2025

Automated FinOps for Cloud Infrastructure Cost Management

Guilherme de Matos Ferreira de Almeida

Mestrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

President: Prof. Ademar Manuel Teixeira de Aguiar

External Examiner: Prof. Hugo Daniel Abreu Peixoto

Supervisor: Prof. André Monteiro de Oliveira Restivo

July 17, 2025

Abstract

In recent years, the adoption of cloud computing has become increasingly popular, with organizations and individuals leveraging its benefits for various applications. With the growth in demand for cloud environments, cloud providers have introduced a vast catalog of services, each with its own pricing model and characteristics. This growth has resulted in a complex ecosystem for cloud tenants, making managing and optimizing cloud costs and resources increasingly challenging. Despite native cost management tools offered by cloud providers, these tools may present a delay of up to 24 hours in reporting, lack transparency, and provide limited granularity, hindering real-time decision-making and possibly leading to unexpected costs.

This dissertation proposes and develops two FinOps frameworks, validated through a User Demand Survey, with the common goal of providing real-time visibility and optimization of cloud costs and resources. The first framework, built on top of Amazon CloudWatch, focuses on Amazon EC2 (Elastic Compute Cloud) instances through a modular and extensible architecture that enables future addition of new modules for other AWS (Amazon Web Services) services. It implements analysis and optimization algorithms that consider resource utilization metrics, such as CPU, network, and memory usage, to generate actionable cost and resource optimization recommendations. The second framework, built on top of OpenCost, is cloud-provider agnostic and operates on Kubernetes clusters, providing detailed cost and resource visibility at cluster, namespace, node, and pod levels. Both frameworks share essential features, including integration with industry-standard tools like Prometheus and Grafana, pre-configured alerts and dashboards, and real-time monitoring capabilities.

Both frameworks were evaluated using real-world scenarios, demonstrating their effectiveness in providing real-time insights and observability. The Kubernetes-based framework predicted costs in real-time with a mean absolute percentage error of 1.46% over twenty days of testing on an Amazon EKS (Elastic Kubernetes Service) cluster. The CloudWatch-based framework was tested using an in-built simulator mode with real-world data exported from Amazon CloudWatch, preventing incurring additional costs. It provided accurate base cost calculations and showed that it can effectively provide real-time visibility into cloud resources.

The results of this work confirm that real-time monitoring systems can provide indicators that lead to measurable cost savings and better resource allocation, validating the central hypothesis of this dissertation. This work contributes to the field of FinOps by offering two easy-to-use, modular, and extensible open-source frameworks that provide real-time visibility and optimization over cloud costs and resources, directly addressing the lack of real-time monitoring and granularity in existing cloud cost management tools.

Keywords: FinOps, Cloud Computing, Kubernetes, AWS, Amazon EC2, Amazon EKS, Real-Time Monitoring, Resource Optimization, Cost Optimization

Resumo

Nos últimos anos, a adoção da computação em nuvem tornou-se cada vez mais popular, com organizações e indivíduos a tirarem partido dos seus benefícios para diversas aplicações. Com o aumento da procura por ambientes na nuvem, os fornecedores de serviços em nuvem introduziram um vasto catálogo de serviços, cada um com o seu próprio modelo de preços e características. Este crescimento resultou num ecossistema complexo para os utilizadores, tornando a gestão e a otimização de custos e recursos na nuvem cada vez mais desafiante. Apesar das ferramentas nativas de gestão de custos oferecidas pelos fornecedores de serviços na nuvem, estas podem apresentar um atraso de até 24 horas na geração de relatórios, falta de transparência e granularidade limitada, dificultando a tomada de decisões em tempo real e podendo levar a custos inesperados.

Esta dissertação propõe e desenvolve duas plataformas de *FinOps*, validadas através de um inquérito, com o objetivo comum de proporcionar visibilidade e otimização em tempo real dos custos e recursos na nuvem. A primeira plataforma, desenvolvida sobre o *Amazon CloudWatch*, foca-se no serviço *Amazon EC2 (Elastic Compute Cloud)* tendo uma arquitetura modular e extensível que permite a futura adição de novos módulos para outros serviços da AWS (*Amazon Web Services*). Implementa algoritmos de análise e otimização que consideram métricas de utilização de recursos, como CPU, rede e memória, para gerar recomendações práticas de otimização de custos e recursos. A segunda plataforma, desenvolvida sobre o *OpenCost*, é agnóstica ao fornecedor de serviços na nuvem e opera em clusters *Kubernetes*, fornecendo visibilidade detalhada dos custos e recursos a nível do *cluster*, *namespace*, nó e *pod*. Ambas as plataformas partilham características essenciais, tais como integração com ferramentas padrão da indústria, como *Prometheus* e *Grafana*, alertas e gráficos pré-configurados, e capacidades de monitorização em tempo real.

Ambas as plataformas foram avaliadas em cenários reais, provando ser eficazes a proporcionar observabilidade em tempo real. A plataforma baseada em *Kubernetes* previu custos em tempo real com um erro percentual absoluto médio de 1.46% ao longo de vinte dias de testes num cluster *Amazon EKS (Elastic Kubernetes Service)*. A plataforma baseada no *CloudWatch* foi testada usando um simulador integrado com dados reais exportados do *Amazon CloudWatch*, evitando custos adicionais. Esta plataforma forneceu cálculos precisos de custos base e demonstrou ser eficaz em proporcionar visibilidade em tempo real sobre os recursos na nuvem.

Os resultados deste trabalho confirmam que sistemas de monitorização em tempo real podem fornecer indicadores que conduzem a poupanças de custos mensuráveis e melhor alocação de recursos, validando a hipótese central desta dissertação. Este trabalho contribui para o campo de *FinOps* ao oferecer duas plataformas de código aberto, fáceis de usar, modulares e extensíveis, que proporcionam visibilidade e otimização em tempo real sobre os custos e recursos na nuvem, respondendo diretamente à falta de monitorização em tempo real e granularidade nas ferramentas de gestão de custos na nuvem.

Palavras-Chave: FinOps, Computação em Nuvem, Kubernetes, AWS, Amazon EC2, Amazon EKS, Monitorização em Tempo Real, Otimização de Recursos, Otimização de Custos

UN Sustainable Development Goals

The United Nations Sustainable Development Goals (SDGs) provide a global framework to achieve a better and more sustainable future for all. It includes 17 goals to address the world's most pressing challenges, including poverty, inequality, climate change, environmental degradation, peace, and justice.

Cloud computing infrastructure consumes significant amounts of energy and computational resources, contributing to a substantial digital carbon footprint globally. FinOps practices and this project tackle these sustainability challenges by reducing resource waste and improving operational efficiency in cloud environments.

The specific Sustainable Development Goals mentioned have the following names:

SDG 7 Ensure access to affordable, reliable, sustainable and modern energy for all.

SDG 8 Promote sustained, inclusive and sustainable economic growth, full and productive employment and decent work for all.

SDG 9 Build resilient infrastructure, promote inclusive and sustainable industrialization and foster innovation.

SDG 12 Ensure sustainable consumption and production patterns.

SDG 13 Take urgent action to combat climate change and its impacts.

SDG	Target	Contribution	Performance Indicators and Metrics
7	7.3	Real-time cost optimization reduces energy waste in cloud data centers by eliminating idle and underutilized resources, contributing to increased energy efficiency.	Percentage reduction in idle or underutilized resources; Energy efficiency scores of optimized cloud resources
8	8.2	The frameworks enable higher productivity through automation of manual cost analysis tasks, reducing operational burden and allowing teams to focus on core activities.	Reduction in manual cost analysis time; Improvement in operational efficiency metrics
	8.4	By promoting efficient resource utilization and reducing cloud waste, the frameworks support decoupling economic growth from environmental degradation through improved resource efficiency.	Percentage reduction in cloud resource waste; Cost savings achieved through optimization recommendations
9	9.4	The open-source, extensible frameworks promote sustainable industrialization by providing accessible tools for efficient cloud resource management across organizations of all sizes.	Number of organizations adopting the frameworks
	9.5	The research contributes to innovation in cloud cost management through novel algorithms for real-time resource monitoring, efficiency scoring, anomaly detection, and predictive cost forecasting.	MAPE accuracy of cost predictions
12	12.2	Real-time monitoring and optimization of cloud resources promotes sustainable management and efficient use of computational resources, reducing digital waste.	Efficiency scores of cloud resource utilization; Reduction in computational resource waste
	12.6	The frameworks encourage organizations to adopt sustainable practices in cloud computing by providing transparency and actionable insights for responsible cloud usage.	Number of organizations implementing sustainable cloud practices; Adoption rate of optimization recommendations
13	13.3	Reducing cloud infrastructure energy consumption through resource optimization contributes to climate change mitigation by decreasing the carbon footprint of data centers.	Reduction in energy consumption of cloud data centers; Carbon footprint reduction

Acknowledgments

First and foremost, I would like to express my gratitude to my supervisor and co-supervisor, Professor André Restivo and João Pedro Dias, for their invaluable guidance and support throughout the various stages of this work. Their expertise and support were essential to the completion of this dissertation.

I would also like to thank Kuehne + Nagel for the opportunity to work on this project, which at first was challenging but ultimately led me to step out of my comfort zone and grow as a professional.

To my family, I owe you a special thank you for your continuous support and encouragement throughout my academic journey. To my father, for the constant support and help in my studies, and for always being there to listen to my ideas and doubts. To my mother, for the constant support and encouragement she has given me throughout my life, and for always believing in me and pushing me to do my best. To my grandparent, for their wisdom and steady presence in my life, which have always been a source of motivation and comfort. To my uncle, for living with me during these last few months and dealing with my sometimes less-than-regular schedules. Your love and support have been a constant in my life, and I am truly grateful for everything you have done for me.

Last but not least, I would like to thank my friends for being there for me during this journey. To the friends I made during these five years, for the moments we shared, the laughs, the late-night study sessions, and the academic life we lived together, which made this journey much more enjoyable and memorable. To my closest friends — they know who they are — thank you for always being there to listen to and support me, even when I was overwhelmed with work and life in general. Your friendship has been a source of strength and comfort, and I wouldn't be who I am today without you.

Guilherme Almeida

*“Beware of little expenses;
a small leak will sink a great ship”*

Benjamin Franklin

Contents

1	Introduction	1
1.1	Context	1
1.2	Motivation	2
1.3	Problem	2
1.4	Research Goals	3
1.5	Document Outline	5
2	Background	6
2.1	Amazon Elastic Compute Cloud	6
2.1.1	EC2 Pricing Models	7
2.1.2	Additional Cost Considerations	7
2.2	Kubernetes Architecture and Core Concepts	8
2.2.1	Fundamental Resources	8
2.2.2	Workload Management Resources	9
2.2.3	Data and Control Planes	10
2.2.4	Configuration Management	11
2.2.5	Storage Management	11
2.2.6	Limits and Requests	12
2.3	Amazon Elastic Kubernetes Service	12
2.3.1	EKS Pricing	12
2.4	Visualization, Alerting, and Monitoring	13
2.4.1	Prometheus and AlertManager	13
2.4.2	Grafana	13
3	State-of-The-Art	14
3.1	Research Domain	15
3.2	Identification Phase	16
3.2.1	Search String Identification	16
3.2.2	Refinement Process	16
3.2.3	Results	18
3.2.4	Search Filters	18
3.2.5	Duplicate Removal	19
3.3	Screening Phase	19
3.3.1	Eligibility Criteria	19
3.3.2	Full-Text Analysis	20
3.4	Results and Discussion	20
3.5	Research Developments	21
3.6	Analysis of Research Developments	25

3.7	Industry Trends and Market Analysis	27
3.7.1	Existing Solutions	27
3.7.2	Analysis of Market Solutions	29
3.8	Summary	30
4	Problem Statement and Methodology	32
4.1	Problem Statement	32
4.2	Hypothesis	33
4.3	Research Questions	33
4.4	Solution Overview	35
4.5	Expected Outcomes	36
4.6	User Demand Survey	37
4.6.1	Survey Overview	38
4.6.2	Key Findings: Validating the Problem Statement	38
4.6.3	Framework Demand and Adoption Potential	39
4.6.4	Implications for Research Development	41
5	CloudWatch-based Framework	42
5.1	Introduction	42
5.2	Architecture Overview	43
5.2.1	Design Principles	43
5.2.2	System Architecture	44
5.2.3	Operational Modes	46
5.2.4	Configuration Management	47
5.2.5	Resource Definition and Limits	48
5.2.6	Resource Weight Distribution	48
5.3	Implementation Details	49
5.3.1	Metric Collection Strategy	49
5.3.2	Alert Configuration and Monitoring	50
5.3.3	Data Storage in Prometheus	50
5.3.4	Algorithms Overview	50
6	Kubernetes-based Framework	58
6.1	Introduction	58
6.2	Architecture Overview	59
6.2.1	Namespace-Level Overview	59
6.2.2	Node-Level Overview	61
6.3	Implementation Details	62
6.3.1	Data Storage and Retention Strategy	62
6.3.2	Resource Waste, Requests and Limits	63
6.3.3	Efficiency Score Calculation	64
6.3.4	Resource Utilization Metrics	64
6.3.5	Rightsizing Recommendations and Optimization Savings	65
6.3.6	Anomaly Detection Methodology	66
6.3.7	Cost Forecasting	68

7	Experimental Setup and Results	70
7.1	CloudWatch-based Framework Experimental Setup	70
7.2	Kubernetes-based Framework Experimental Setup	71
7.2.1	Kind Cluster Configuration	71
7.2.2	EKS Cluster Configuration	72
7.3	Validation Methodology	74
7.3.1	Cost Prediction Accuracy Criteria	74
7.3.2	User Demand Evaluation Criteria	75
7.4	Cost Prediction Accuracy Results	76
7.4.1	Kubernetes-based Framework Cost Prediction	76
7.4.2	CloudWatch-based Framework Sanity Check	78
8	Conclusions	80
8.1	Research Questions	81
8.2	Hypothesis Revisited	83
8.3	Thesis Validation	83
8.3.1	Kubernetes-based Framework Cost Prediction	83
8.3.2	CloudWatch-based Framework Sanity Check	84
8.3.3	Reviewing user demands taking into account technical details	85
8.4	Threats to Validity	87
8.5	Summary of Contributions	88
8.6	Limitations and Challenges	89
8.7	Future Work	89
	References	91
A	Selected Studies	98
B	Implementation Details	101
B.1	Kubernetes-based Framework	102
B.1.1	Pseudocode Snippets	102
B.1.2	Kind Deployment Script	103
B.2	CloudWatch-based Framework	107
B.2.1	Pseudocode Snippets	107
B.2.2	Configuration Files	109
C	Grafana Dashboards	111
C.1	Kubernetes-based Framework	111
C.1.1	Dashboards - OpenCost / Overview	111
C.1.2	Dashboards - OpenCost / Namespace	112
C.1.3	Dashboards - OpenCost / Node	114
C.1.4	Dashboards - Specific Scenarios	115
C.2	CloudWatch-based Framework	116
D	Kubernetes-based Framework Monitoring Metrics	119

E	Alert Configurations	121
E.1	Kubernetes-based Framework	121
E.1.1	Cost-Related Alerts	121
E.1.2	Optimization Recommendation Alerts	125
E.1.3	Resource-Level Alerts	126
E.2	CloudWatch-based Framework	130
E.2.1	Cost-Related Alerts	130
E.2.2	Resource Utilization and Efficiency Alerts	131
E.2.3	Optimization Recommendation Alerts	133
E.2.4	Trend Analysis Alerts	134
F	User Demand Survey	136
F.1	Survey Questions	136
F.2	Survey Results	147
F.2.1	Respondent Demographics and Background	147
F.2.2	Current Cost Management Practices	147
F.2.3	CloudWatch-based Framework Evaluation	149
F.2.4	Kubernetes-based Framework Evaluation	150
F.2.5	Implementation and Adoption Factors	151

List of Figures

2.1	Components of Kubernetes architecture (source: [52])	8
2.2	Node, Pod, and Namespace relationship in Kubernetes	9
3.1	PRISMA Flowchart	15
3.2	Number of papers published per year	20
4.1	Cost impact versus visibility gap	38
4.2	Primary cloud cost management challenges	39
4.3	Framework adoption likelihood	40
4.4	Framework cost reduction expectations	40
5.1	CloudWatch-based Framework Architecture Overview	44
6.1	Kubernetes-based Framework Architecture Overview - Namespace Level	59
6.2	Kubernetes-based Framework Architecture Overview - Node Level	62
7.1	Opencost UI Integration with AWS Athena	73
C.1	Pod and Container Summary dashboard	111
C.2	FinOps Middleware - Total Potential Savings view	112
C.3	Cloud Resources dashboard	112
C.4	Namespace Summary dashboard	112
C.5	FinOps Middleware dashboard	113
C.6	Summary dashboard	113
C.7	Pod Summary dashboard	113
C.8	Container Summary dashboard	114
C.9	PV Summary dashboard	114
C.10	Node Resource Utilization dashboard	115
C.11	Cluster Summary - Infrastructure Change Impact	115
C.12	EC2 Cost Analysis Dashboard	116
C.13	EC2 Resource Utilization Dashboard	117
C.14	EC2 Alerting Dashboard	118

List of Tables

3.1	Initial Search Strings	16
3.2	Refined Search Strings	18
3.3	Search String Results for Each Database	18
3.4	Inclusion and Exclusion Criteria for Literature Selection	19
3.5	SoTA literature and their characteristics	26
3.6	Industry solutions and their characteristics	30
7.1	Interpretation of typical MAPE values	75
7.2	Cloud cost data	77
A.1	Selected Studies (2018-2020)	98
A.2	Selected Studies (2021)	98
A.3	Selected Studies (2022)	99
A.4	Selected Studies (2023)	99
A.5	Selected Studies (2024)	100
D.1	Node Exporter Metrics	119
D.2	cAdvisor/Kubelet Metrics	119
D.3	OpenCost Metrics	120
D.4	kube-state-metrics Metrics	120
F.1	Survey respondent demographics	147
F.2	Cost decision impact and visibility assessment	148
F.3	Monthly time investment in cost analysis	148
F.4	Cloud cost management challenges	149
F.5	Current optimization identification processes	149
F.6	CloudWatch-based framework feature importance ratings	150
F.7	Kubernetes cost monitoring approaches	150
F.8	Kubernetes cost management pain points	150
F.9	Kubernetes framework feature importance ratings	151
F.10	Valuable cost optimization alert types	151
F.11	Cost visibility granularity preferences	151
F.12	Deployment importance and implementation barriers	152
F.13	Success criteria and cost reduction thresholds	152
F.14	Expected benefits and adoption likelihood	153

List of Acronyms

AWS	Amazon Web Services	1
EC2	Elastic Compute Cloud	6
EKS	Elastic Kubernetes Service	6
RDS	Relational Database Service	28
GKE	Google Kubernetes Engine	58
AKS	Azure Kubernetes Service	58
CUR	Cost and Usage Reports	32
CSP	Cloud Service Provider	1
FinOps	Financial Operations	2
GCP	Google Cloud Platform	1
IaaS	Infrastructure as a Service	1
PaaS	Platform as a Service	1
SaaS	Software as a Service	1
TWh	Terawatt-hour	2
VM	Virtual Machine	22
PID	Proportional-Integral-Derivative	24
MILP	Mixed-Integer Linear Programming	21
QoS	Quality of Service	22
SLA	Service Level Agreement	47
ML	Machine Learning	24
AI	Artificial Intelligence	24
ECDF	Empirical Cumulative Distribution Function	21
CI/CD	Continuous Integration/Continuous Deployment	28
ROI	Return on Investment	27
SME	Small and Medium-sized Enterprise	28
API	Application Programming Interface	43
IaC	Infrastructure as Code	23
CWP	Cloud Waste Point	22
CUS	Cloud Usage Score	22
NIST	National Institute of Standards and Technology	25

MAPE	Mean Absolute Percentage Error	74
IT	Information Technology	76
SQL	Structured Query Language	72
PV	Persistent Volume	11
PVC	Persistent Volume Claim	11
ECR	Elastic Container Registry	73
EBS	Elastic Block Store	7
AZ	Availability Zone	7
ELB	Elastic Load Balancer	68
HPC	High Performance Computing	6
GB	Gigabyte	55
WAL	Write-Ahead Logging	13
PromQL	Prometheus Query Language	60
OPEX	Operational Expenses	85

Chapter 1

Introduction

This chapter outlines the theme of this dissertation, as well as its context, problems, and goals.

Section 1.1 introduces the reader to the background where this dissertation is inserted. Section 1.2 goes further into detail about the motivation of this research. Section 1.3 outlines the issues present at the core of this work. Section 1.4 presents the questions that will guide this research and formulates the hypothesis that will be tested during this dissertation. Finally, Section 1.5 provides an overview of the document structure, outlining the content of each chapter.

1.1	Context	1
1.2	Motivation	2
1.3	Problem	2
1.4	Research Goals	3
1.5	Document Outline	5

1.1 Context

Cloud computing is a paradigm that allows tenants — individuals or organizations who share a common Cloud Service Provider (CSP), where each operates within a single, isolated environment while using a shared infrastructure — to provision resources, such as applications, networks, infrastructure, and storage, with minimal effort [59].

Although the term cloud is commonly associated with Public Clouds, such as Amazon Web Services (AWS) or Google Cloud Platform (GCP), cloud computing can be segmented into four deployment models: (1) Private Cloud, (2) Public Cloud, (3) Hybrid Cloud, and (4) Community Cloud.¹[59, 81].

CSPs provide services based on three main service models: (1) Software as a Service (SaaS), (2) Platform as a Service (PaaS), and (3) Infrastructure as a Service (IaaS), each providing a different level of flexibility, responsibilities, and control over the resources [59].

¹In this dissertation, unless otherwise specified, references to the term "cloud" implicitly refer to Public Cloud deployments.

As the demand for cloud computing grows, various **CSPs** have expanded their services ecosystem, making it increasingly appealing for tenants to shift from on-premises environments to the cloud. However, with this shift, tenant awareness increased, and financial management became a critical factor for tenant satisfaction. Tenants want to make the most cost-effective decisions and require transparency over where their money is spent, enabling them to make real-time adjustments.

In response to these concerns, Financial Operations (**FinOps**) emerged, introducing management processes and practices that enable financial accountability to the highly dynamic cost model of cloud [16, 47].

Despite the availability of cost-reporting tools in various cloud environments and the introduction of **FinOps**, these mechanisms often involve delay in reporting, affecting real-time decision-making and potentially leading to additional costs.

This dissertation was proposed by Kuehne + Nagel, a global logistics company based in Switzerland.

1.2 Motivation

Managing cloud costs is crucial to ensure sustainable cloud usage; however, despite the existence of cost management tools, such as AWS Cost Explorer, there is still a lack of transparency on the origin of these costs. In addition, these reports might appear with a delay of up to 24 hours [15, 72], making it possible to incur unwanted costs due to the lack of real-time knowledge.

This may lead to situations where unwanted resources are running and not only add up to enormous bills but also increase the tenant's carbon footprint. These inefficiencies contribute to an alarming growth of energy consumption among data centers, which reached an estimated amount of 460 Terawatt-hour (**TWh**) in 2022 and a predicted consumption of more than 1000 **TWh** in 2026 — equivalent to the total power consumption of Japan. [1, 2]

While progress has already been made within some ecosystems, such as Kubernetes [49], there is still a lack of solutions that work for various services and tackle these issues.

This dissertation proposes the creation of a **FinOps** system that enables tenants to monitor and analyze costs in real-time, automatically triggering alerts and recommending actions based on user-defined policies. This system aims to reduce both cloud costs and improve resource allocation, all while reducing operational burden.

1.3 Problem

Cloud-native cost detection solutions can delay report generation by up to 24 hours [15, 72]. This hinders proactive decision-making and could lead tenants to incur additional costs due to unwanted resource creation, over-allocation, or even forgetting to deallocate resources when not in use.

Furthermore, the way these reports are displayed is not as transparent and granular as desirable, requiring a lot of manual effort and financial expertise to interpret. This also means tenants must

spend additional time manually analyzing reports, which may impact financial management and distract them from their core activities, increasing operational burden.

Additionally, pricing models are often complex and dependent on the CSPs and service-specific characteristics [58]. This adds an additional layer of complexity, making it even more challenging to estimate, compare, and optimize cloud expenses. Even with the proper expertise, this process is very prone to error.

1.4 Research Goals

To achieve the goals of this dissertation and effectively address the problems outlined in Section 1.3, it is essential to know how to retrieve data from the cloud in real-time, which algorithms and heuristics can be used to achieve our goals, and which actions should be recommended so tenants don't incur additional costs. With that in mind, the following questions can be formulated:

RQ1: Which tools can be used to retrieve data from cloud environments in real-time?

The review of the available state-of-the-art allows a deeper understanding of the current landscape of cloud cost management tools and their capabilities. This allows the identification of the most suitable tools for real-time data retrieval from cloud environments, as well as their limitations, capabilities, and requirements, such as licensing, possible costs, and CSPs support.

RQ2: Which algorithms and heuristics should be developed to catalog cloud resources, monitor costs, and optimize resource allocation?

Similarly to RQ1, it is possible to identify algorithms and heuristics that can be used or improved to efficiently catalog cloud resources, monitor and optimize costs, and improve resource allocation by reviewing the state-of-the-art. Despite the lack of artifacts — algorithms, pseudocode, GitHub repositories, etc. — in the literature, it is still possible to use the core concepts and ideas presented to develop algorithms and heuristics that can be used in this system. Understanding which metrics should be considered when monitoring costs and resource allocation, and whether historical cloud cost data is necessary for the algorithms to function correctly, is essential. on this specific

RQ3: Which actions or policies can be recommended to reduce cloud costs, improve resource efficiency, and enable more informed decision-making?

It is essential to understand which actions or policies can be recommended to tenants so that they can effectively reduce cloud costs and improve resource allocation. This includes understanding how policies can be tailored to different services' cost models, how forecasting based on previous usage can affect these recommendations, and to what extent these recommendations can be automated. Furthermore, the User Demand Survey Analysis presented in Sections 4.6.1 and 8.3.3 provide insights into the actions and policies tenants would like to see implemented in the system.

RQ4: How do real-time cost monitoring and actionable recommendations improve a tenant's financial operations compared to cost reports?

This question aims to understand the impact of real-time cost monitoring and actionable recommendations on a tenant's financial operations compared to traditional cost reports. Understanding the benefits a tenant can expect from a system that provides these features, such as cost savings and reduced operational expenses, is essential. The knowledge obtained during the development stage of this system will provide direct insights into this topic.

RQ5: How can the system be integrated into existing environments?

Integration is a crucial aspect of any system, and it is essential to understand how the proposed solution can be integrated into existing environments. This includes understanding the requirements for integration and how the system can be made compatible with existing tools and workflows. Similarly to **RQ4**, the development stage provides insights into the system integration process, including the architectural decisions made to ensure compatibility with existing environments and the ease of deployment.

This dissertation aims to develop a solution that enables tenants to monitor, analyze, and optimize their cloud costs and resources in real-time while providing actionable recommendations to reduce these costs, improve resource allocation, and prevent cost overruns.

Having these research questions in mind, the central hypothesis of this dissertation is formulated as follows:

H: *A system that monitors and forecasts costs in real-time can provide decision-makers with indicators that lead to measurable cost savings and better resource allocation.*

A system that monitors and forecasts costs in real time refers to a system that can gather real-time data from a cloud ecosystem and predict future costs based on historical usage data.

Decision-makers refer to cloud administrators or tenants who have the authorization to manage cloud resources and access to cost data.

Indicators that lead to measurable cost savings and better resource allocation imply that the cloud costs using this system are less — assuming a non-optimized or imperfect setup — than the costs using previous cost management solutions on an equivalent time frame after the recommendations made by the system are applied.

The research goals — research questions and hypothesis — presented in this Section will be further discussed in Chapter 4, where this dissertation's problem statement and methodology are presented.

1.5 Document Outline

Apart from this chapter, the document contains seven additional chapters. Chapter 2 provides the necessary background information to understand the concepts and technologies used throughout this dissertation. Chapter 3 reviews the latest literature on **FinOps** and cloud cost management, providing an overview of the state-of-the-art in this field. Chapter 4 presents the main problem this dissertation aims to address, details the research questions and central hypothesis of this dissertation, and the methodology and approach to develop the proposed solution, backed by a user demand survey analysis. Chapter 5 presents the design and implementation of the **FinOps** framework based on Amazon Cloudwatch, including the architecture, components, and algorithms used to monitor and optimize costs. Chapter 6 presents the design and implementation of the **FinOps** framework for Kubernetes environments, including the architecture, components, and algorithms used to monitor and optimize costs. Chapter 7 outlines the experimental setup used to validate the frameworks presented in this dissertation, defines the methodology used to evaluate the results, and presents the results obtained from the experiments. Finally, Chapter 8 answers the research questions, evaluates the hypothesis, discusses the findings from the experiments and the user demand survey, presents the threats to validity of these findings, gives an overview of the contributions made throughout this dissertation, discusses the limitations and challenges faced during this work, and outlines potential directions for future research and development on this topic.

Chapter 2

Background

This chapter introduces concepts and technologies that are mentioned during this dissertation. It provides a foundation for understanding the research domain and the rest of this dissertation.

Section 2.1 introduces Amazon Elastic Compute Cloud (EC2), and details its pricing models and additional cost considerations. Section 2.2 introduces Kubernetes, providing an overview of its architecture and core concepts. Section 2.3 introduces Amazon Elastic Kubernetes Service (EKS), its features, and how it is priced. Finally, Section 2.4 introduces Prometheus, Grafana, and AlertManager.

2.1	Amazon Elastic Compute Cloud	6
2.2	Kubernetes Architecture and Core Concepts	8
2.3	Amazon Elastic Kubernetes Service	12
2.4	Visualization, Alerting, and Monitoring	13

2.1 Amazon Elastic Compute Cloud

Amazon Elastic Compute Cloud [5], commonly referred to as Amazon EC2, is a compute platform that provides scalable and resizable compute capacity in the cloud. This service is one of the core components of AWS and is designed to offer users the ability to use virtual machines, known as instances, to run any workload. These instances are organized into families optimized for different workload characteristics, such as compute, memory, or storage. General-purpose instances balance compute, memory, and networking resources. Compute optimized instances provide high-performance processors for CPU-intensive tasks. Memory-optimized instances are designed to deliver high memory performance for memory-intensive applications, such as in-memory databases or real-time big data analytics. Storage-optimized instances offer high disk throughput and are ideal for data warehousing and log processing scenarios. Less common instance types include High Performance Computing (HPC) optimized instances, and accelerated computing instances [21].

2.1.1 EC2 Pricing Models

Amazon **EC2** offers multiple pricing models that significantly affect the cost of running instances. Among these models, the most common are:

On-Demand: This model charges users based on the actual usage of instances, charging by the hour or second, with no long-term commitments. While this model is the most flexible, it is also the most expensive variant.

Reserved instances: This model offers substantial discounts relative to on-demand pricing, ranging from 31% to 72% depending on the term length. The rationale is that users reserve instances for a one or three-year term, committing to use the instance for that period. Payment can be made in various ways, including all upfront, partial upfront, and no upfront [68].

Spot instances: This model allows users to bid on unused **EC2** capacity, having the potential to save up to 90% compared to on-demand pricing [6]. However, spot instances may be interrupted by **AWS** with a two-minute warning if **AWS** needs the capacity back [82]. This means that spot instances can be very cost-effective, but they are not suitable for all workloads, especially those requiring high availability or that cannot tolerate interruptions.

Savings Plans: This flexible pricing model offers significant savings on **EC2** usage in exchange for a commitment to use a specific amount of compute power for a one or three-year term. Savings Plans provide a more flexible alternative to reserved instances, allowing users to apply their discount across any **EC2** instance type, region, operating system, and tenancy. There are two types of savings plans: (1) compute savings plan, which provides the most flexibility and applies to any **EC2** instance regardless of region, instance family, operating system, or tenancy, allowing users to save up to 66% compared to on-demand pricing; and (2) instance savings plan, which applies to a specific instance family within a region, providing savings of up to 72% compared to on-demand pricing. Savings plans also apply to other **AWS** services, namely Fargate and Lambda [23].

This dissertation focuses on the on-demand pricing model, which is the most flexible and commonly used pricing model in **AWS**.

2.1.2 Additional Cost Considerations

In addition to the instance pricing models, several factors can significantly affect the overall cost of running **EC2** instances. These factors include: (1) data transfer charges, which apply for traffic moving between regions, Availability Zones (**AZs**), or to and from the internet; (2) Elastic Block Store (**EBS**) volumes for persistent storage, with costs determined by factors such as the volume type, size, provisioned performance levels, and the amount of data transferred; and (3) premium features such as enhanced networking or dedicated tenancy.

The combination of instance variants, pricing models, and additional services creates a very complex pricing structure, making it difficult for users to estimate costs accurately.

2.2 Kubernetes Architecture and Core Concepts

Kubernetes is an open-source platform designed to orchestrate the deployment, scaling, and management of containerized applications across clusters of machines through a master-worker architecture.

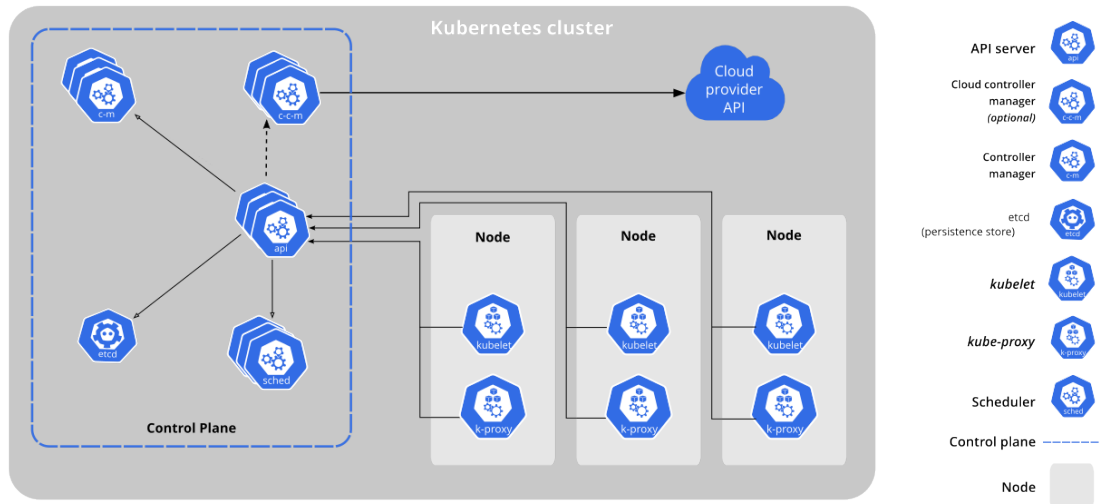


Figure 2.1: Components of Kubernetes architecture (source: [52])

2.2.1 Fundamental Resources

Nodes represent individual machines in the cluster, whether physical or virtual machine instances.

On each node, Kubernetes runs a set of components that are essential for the cluster's operation, as illustrated in Figure 2.1. Among these components, the following can be highlighted:

Kubelet: Kubelet is an agent that runs on every node and communicates with the control plane — explained in Section 2.2.3 — and is responsible for: (1) registering the node within the cluster; (2) monitoring pods and containers on that node; (3) ensuring containers are running as specified; (4) communicating with the Kubernetes API server; and (5) enforcing the declared pod specifications.

Kube-proxy: Kube-proxy is a network proxy that runs on each node and is responsible for: (1) maintaining network rules for communication between pods; (2) implementing service abstractions by routing traffic to appropriate pods based on service definitions; (3) load balancing requests across multiple pod replicas; and (4) managing network connectivity between services and external traffic.

Container Runtime: The container runtime is the software responsible for running containers on the node, such as Docker, containerd, or CRI-O.

Nodes provide the computational resources and infrastructure required to support workloads in a Kubernetes cluster. Each node serves as the host for one or more pods.

Pods are the smallest deployable units in Kubernetes, containing one or more tightly coupled containers that share storage and network resources. Containers within a pod can communicate with each other using localhost and share mounted volumes for persistent storage. Pods represent ephemeral units that can be created, destroyed, and recreated as needed by the Kubernetes control plane, ensuring that the system remains in the desired state.

Namespaces provide logical separation within a cluster, enabling multiple teams or applications to coexist within the same cluster without interfering with each other. Namespaces allow for resource isolation, access control, and organization of resources. Each namespace keeps separate resource quotas [69], access control policies, and naming scopes, meaning that names within a namespace must be unique. Still, the same name can be used across different namespaces.

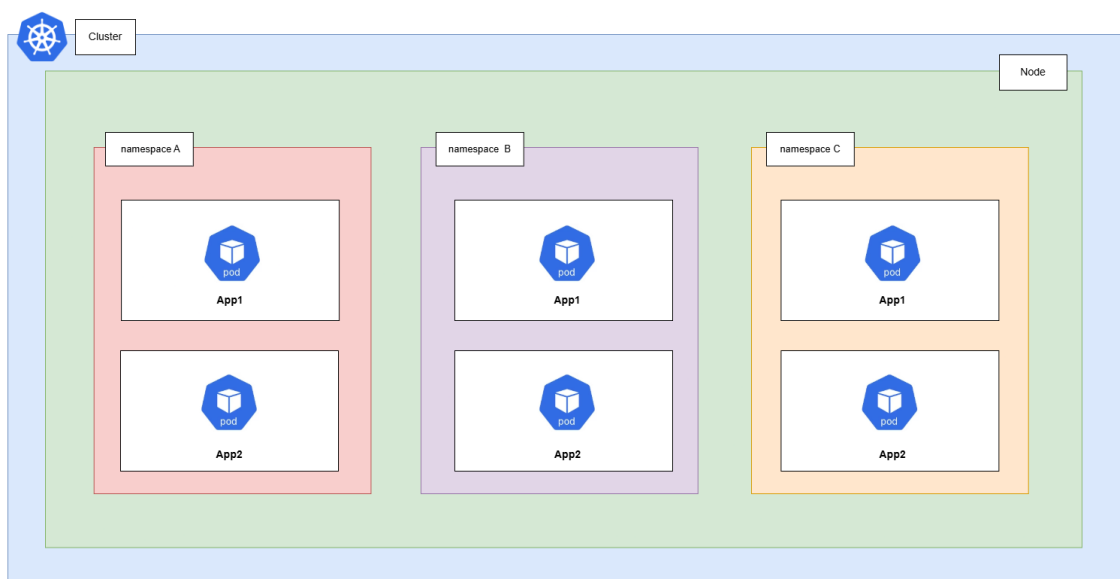


Figure 2.2: Node, Pod, and Namespace relationship in Kubernetes

Figure 2.2 illustrates the relationship between nodes, pods, and namespaces in Kubernetes. It is a simplified representation of how these fundamental resources are organized within a Kubernetes cluster; however, scenarios in production clusters are often more complex, with multiple pods running on each namespace, and the possibility of having multiple nodes across a single namespace.

2.2.2 Workload Management Resources

While pods represent the smallest deployable units in Kubernetes, they are typically not created directly in production environments. Instead, Kubernetes provides higher-level abstractions for pods and their lifecycles:

Deployments: Deployments provide declarative management of pods, allowing users to define the desired state of the pod, such as the number of replicas, the image to use, and the resource requirements. Deployments handle stateless applications with capabilities such

as rolling updates, rollbacks, and replica management. They also ensure that the specified number of replicas remain running and automatically replace failed pods.

StatefulSets: StatefulSets manage stateful applications requiring stable network identities, persistent storage, and ordered deployment patterns. Unlike deployments, StatefulSets provide each pod with a unique identifier that remains consistent across rescheduling events. They ensure pods are created and deleted in a specific order, being particularly useful for workloads such as databases or distributed systems.

Services: Services provide a mechanism for exposing applications running in pods to other pods or external clients. They define a stable network endpoint for accessing pods, abstracting the underlying pod IP addresses. Services can be of different types, such as: (1) ClusterIP, which exposes the service on a cluster internal IP address, making it accessible only within the cluster; (2) NodePort, which exposes the service on a static port on each node's IP address, allowing external access to the service; and (3) LoadBalancer, which provisions an external load balancer to distribute the traffic to the service.

ReplicaSets: ReplicaSets ensure that a specified number of pod replicas are running at any given time. While they are often used indirectly through deployments, they can also be used directly to manage a set of pods with the same specification.

DaemonSets: DaemonSets ensure that specific pods run on all or a subset of nodes in the cluster. They are typically used for system-level services that need to run on every node, such as logging and monitoring agents. DaemonSet automatically schedules a pod on the new node when a new node is added to the cluster.

These workload management resources provide a higher level of abstraction for managing pods and their lifecycles, allowing users to define the desired state of their applications and ensuring that the system remains in that state.

2.2.3 Data and Control Planes

The Kubernetes architecture is divided into two main planes: the control and data planes.

The control plane contains the master nodes that manage the cluster's state and orchestrate the deployment and scaling of applications. It serves as the cluster's management layer, responsible for making global decisions about the cluster's state and responding to events in the cluster. As illustrated in Figure 2.1, the control plane consists of several key components that work together to maintain the desired cluster state:

API Server: The API Server serves as the primary interface for interacting with the cluster, exposing the Kubernetes API and serving as the frontend for the control plane.

Scheduler: The Scheduler is responsible for assigning pods to nodes based on resource availability and constraints, ensuring that workloads are distributed efficiently across the cluster.

Controller Manager: The Controller Manager runs various controllers that regulate the state of the cluster by monitoring changes and taking corrective actions when it deviates from the desired state.

Cloud Controller Manager: The Cloud Controller Manager is an optional component that integrates with cloud provider APIs to manage cloud-specific resources such as load balancers, storage volumes, and network routing.

etcd: etcd is a distributed key-value store that serves as the cluster's persistent storage, storing all configuration data, state information, and metadata.

The data plane contains the worker nodes where actual workloads are executed. Each worker node runs kubelet, kube-proxy, and the container runtime, as described in Section 2.2.1. The data plane is responsible for running pods, maintaining network connectivity, and ensuring that containers remain healthy and responsive according to the specifications defined in the control plane.

The separation between the control and data planes provides various benefits. The control plane can focus on orchestration and decision-making without the overhead of running workloads, while the data plane can focus its resources on executing applications. This separation ensures horizontal scalability, as more nodes can be added to the data plane to handle increased workloads without affecting the control plane's performance.

2.2.4 Configuration Management

Kubernetes provides several resources for managing configuration and sensitive data within the cluster.

ConfigMaps store non-sensitive configuration data in a key-value pair format, allowing users to decouple configuration from the application code. ConfigMaps are consumed by pods as environment variables, command-line arguments, or configuration files mounted as volumes.

Secrets are similar to ConfigMaps, but are specifically designed to store sensitive information such as passwords and tokens, since they are encoded and can be encrypted.

2.2.5 Storage Management

For storage management, Kubernetes uses Persistent Volumes (PVs), which represent cluster-wide storage resources, and Persistent Volume Claims (PVCs), which represent storage requests by pods. This abstraction allows pods to request storage resources without knowledge of the underlying infrastructure. When a pod requests storage, Kubernetes binds the PVC to a suitable PV, ensuring the pod can access the required storage resources. PVs can be backed by various storage systems, including local disks, network-attached storage, or even storage services provided by CSPs. This is the case on AWS, where created PVs are translated into EBS volumes.

Storage persists beyond the lifecycle of individual pods, requiring separate monitoring and management.

2.2.6 Limits and Requests

Resource management in Kubernetes is achieved by defining resource requests and limits on a per-container basis.

Resource requests specify the minimum amount of CPU and memory required by a container and influence scheduling decisions by ensuring that pods are scheduled on nodes with sufficient resources.

Resource limits define the maximum amount of CPU and memory a container can use, preventing it from consuming excessive resources and affecting other pods on the same node. Proper configurations of requests and limits are essential for efficient resource utilization and to ensure performance stability.

When limits are set but requests are not, Kubernetes will automatically set the request to be the same as the limit value.

Furthermore, LimitRanges are a policy object that can be used to set default, minimum, and maximum resource requests and limits for pods within a specified namespace. This is useful for (1) automatically applying default resource requests and limits to pods or containers that do not specify them; (2) enforcing minimum and maximum resource utilization per pod or container; and (3) preventing a container from consuming too many or too few resources.

2.3 Amazon Elastic Kubernetes Service

Amazon Elastic Kubernetes Services [57], or Amazon **EKS**, provides a managed Kubernetes service where **AWS** takes care of the control plane, while users are responsible for managing the worker nodes. This service is designed to simplify the deployment, management, and scaling of containerized applications.

2.3.1 EKS Pricing

Control plane costs for the standard **EKS** version are charged at a fixed rate of \$0.10 per hour, translating to approximately \$73 monthly per cluster, regardless of the cluster size or utilization. **AWS** also offers an Extended Support version of **EKS**, which is priced at \$0.60 per hour, leading to an approximately monthly cost of \$438. The main difference between these two versions lies in the extended maintenance period and ongoing critical updates provided for older Kubernetes versions in the Extended Support version [88].

Typically, worker nodes in **EKS** clusters are provisioned as **EC2** instances, which means that the same cost considerations from Sections 2.1.1 and 2.1.2 apply. There are less common alternatives other than using **EC2** instances as worker nodes, such as **EKS** hybrid nodes, which allow users to run **EKS** clusters on their on-premises infrastructure [7], or using Amazon Fargate [71], which is a serverless compute engine for containers that allows users to run containers without managing the underlying infrastructure. For this dissertation, the focus is on **EKS** clusters using **EC2** instances as worker nodes, as this is the most common configuration.

Additional **EKS** costs arise from integrated **AWS** services, such as applicational load balancers, network load balancers, **EBS** volumes for persistent storage, and data transfer charges for cross-region or cross-**AZ** traffic. These costs can vary significantly based on the cluster's specific configuration and usage patterns. They may represent a significant portion of the overall cost of running an **EKS** cluster.

2.4 Visualization, Alerting, and Monitoring

2.4.1 Prometheus and AlertManager

Prometheus [67] is an open-source time-series database and monitoring system that collects metrics through a pull-based model, where it scrapes metrics from configured endpoints at specified intervals. Metrics are stored with a timestamp and dimensional labels, enabling powerful querying and aggregation capabilities. The system organizes data in two-hour blocks containing chunks, indices, and metadata with Write-Ahead Logging (**WAL**), ensuring that the system can recover from failures and maintain data integrity. Samples are stored in a compressed format, typically ranging from 1 to 2 bytes per sample [85].

PromQL, or Prometheus Query Language, is the query language used to retrieve, manipulate, and analyze time series data, supporting mathematical operations, aggregation functions, filtering by labels, and many other features. This capability allows users to create complex queries to extract meaningful insights from the collected metrics.

AlertManager, which is part of the Prometheus ecosystem, is responsible for managing alerts generated by Prometheus expressions. It allows users to define alerting rules based on PromQL expressions, which are evaluated with the periodicity specified in the rule. When an alert condition is met, AlertManager can be configured to send notifications to configured receivers, such as email, Slack, or PagerDuty.

2.4.2 Grafana

Grafana [41] is an open-source visualization and analytics platform that integrates with various data sources, including Prometheus. It provides a user-friendly interface for creating interactive dashboards and visualizing metrics. Grafana supports various visualization options, such as graphs, tables, heatmaps, and more, allowing users to customize their dashboards to suit their needs. Grafana's integration with Prometheus enables users to create real-time visualizations of metrics collected by Prometheus, making monitoring and analyzing the performance of applications and infrastructure easier. Furthermore, it also provides an interface for managing and observing alert rules defined in AlertManager, allowing users to visualize the status of alerts and take appropriate actions.

Chapter 3

State-of-The-Art

This chapter conducts a systematic review following the PRISMA (Preferred Reporting Items for Systematic Reviews and Meta-Analyses) Methodology [65] and presents a State-of-the-Art analysis in FinOps and Cloud Cost Management. It serves as a foundation for the research conducted in this dissertation.

Section 3.1 defines the research domain for this literature review as long as its motivation. Section 3.2 outlines the review methodology, including database selection, search string formulation, refinement, and the initial query results. Section 3.3 describes the criteria used for screening and assessing the eligibility of studies to ensure relevance to this dissertation. Section 3.4 analyzes the results of the previous two sections and presents the motivation and relevance of the following research. Section 3.5 provides an overview of the latest research developments in this field. Section 3.6 groups the literature into categories and analyzes the results. Section 3.7 explores current market trends, existing solutions, and challenges and groups them into suitable categories. Finally, Section 3.8 summarizes the main findings and highlights the identified gaps.

3.1	Research Domain	15
3.2	Identification Phase	16
3.3	Screening Phase	19
3.4	Results and Discussion	20
3.5	Research Developments	21
3.6	Analysis of Research Developments	25
3.7	Industry Trends and Market Analysis	27
3.8	Summary	30

3.1 Research Domain

The rapid adoption of cloud computing has intensified the need for effective cost management. **FinOps** provides a framework for improving financial accountability and optimizing cloud costs, addressing the challenges associated with tracking, allocating, and optimizing cloud expenses.

Despite the current efforts in this area, gaps remain in the literature regarding real-time cost monitoring, management and optimization, and existing **FinOps** implementations in cloud environments. By examining the current state of research, this review aims to highlight these gaps and set the stage for addressing the research questions presented in Section 4.3.

This literature review aims to identify tools and methodologies that enable real-time cost monitoring, resource discovery, resource optimization, and existing **FinOps** implementations in cloud environments. Fig 3.1 contains an overview of the results obtained from the processes conducted during this systematic review. These will be explained in more detail in the following chapters.

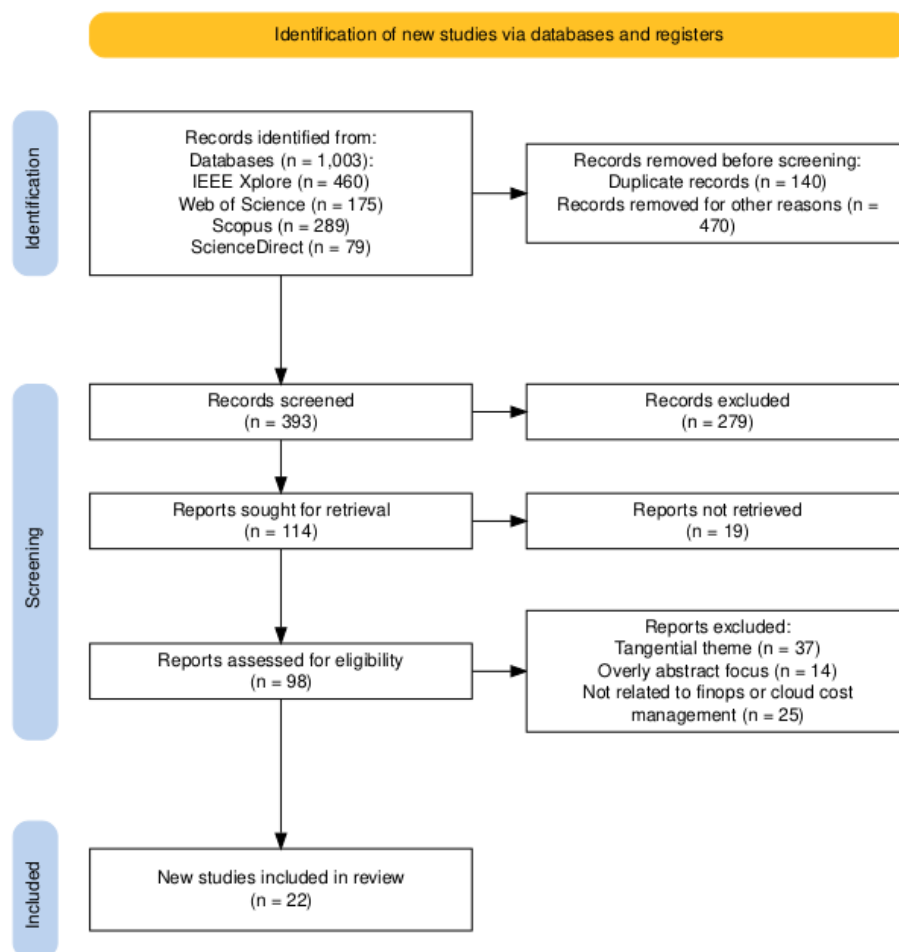


Figure 3.1: PRISMA Flowchart

3.2 Identification Phase

A structured search was conducted across multiple scientific databases, including: (1) Web of Science, (2) IEEE Xplore, (3) Scopus, and (4) ScienceDirect.

3.2.1 Search String Identification

Based on the research questions presented in Section 4.3, four search strings were formulated to capture relevant studies on FinOps, cost optimization, and real-time monitoring. Table 3.1 showcases the initial search strings used. Since each database might use different operators — as is the case with the proximity operator in Web of Science — one of the possible syntax variations was selected; however, the reader should be careful with this caveat when placing the strings on a database.

Table 3.1: Initial Search Strings

Identifier	Search String
QS1	(finops OR cloud) AND (real-time OR "real time") NEAR/2 ("cost optimization" OR "cost management")
QS2	(finops OR cloud) AND cost AND (automated OR policy OR alert) NEAR/2 ("recommendation engine" OR "recommendation system")
QS3	cloud AND (automated OR automatic) NEAR/2 ("resource discovery" OR "resource detection")
QS4	(finops OR "cloud cost") AND ("real time" OR real-time)

3.2.2 Refinement Process

Upon the initial definition of the search strings and verification of some of their limitations, they were iteratively refined until more relevant results were obtained. In this process, the following steps were taken:

Initial Refinements. Two improvements can be made to multiple queries before refining each individually: (1) Adding a dedicated search string with the term *finops* instead of including it as an optional term in multiple strings. Since **FinOps** is a central term in this dissertation and a recent theme, all results from this query were included for further filtering. (2) Replacing the term *cloud* by the composite term *cloud computing* to reduce the number of false positives. After applying these two steps to the initial search strings, they would be the following:

QS1: "cloud computing" AND (real-time OR "real time") NEAR/2 ("cost optimization" OR "cost management")

QS2: "cloud computing" AND cost AND (automated OR policy OR alert) NEAR/2 ("recommendation engine" OR "recommendation system")

QS3: "cloud computing" AND (automated OR automatic) NEAR/2 ("resource discovery" OR "resource detection")

QS4: ("cloud cost" OR "cloud computing cost") AND ("real time" OR real-time)

QS5: finops

QS1 Refinement. The search string *"cloud computing" AND (real-time OR "real time") NEAR/2 ("cost optimization" OR "cost management")* performed poorly, due to being too specific, so it was generalized by removing the proximity operator, which resulted in the query *"cloud computing" AND (real-time OR "real time") AND ("cost optimization" OR "cost management")*.

QS2 Refinement. The search string *"cloud computing" AND cost AND (automated OR policy OR alert) NEAR/2 ("recommendation engine" OR "recommendation system")*, as QS1, retrieved few results. That was tackled by replacing the proximity operator with an AND operator and by adding the terms *"decision-support"* and *"optimization system"*. Furthermore, these terms are widely used in medicine, and the term *NOT medic** was explicitly added. This process resulted in the search string *"cloud computing" AND cost AND (automated OR policy OR alert) AND ("recommendation engine" OR "recommendation system" OR decision-support OR "optimization system") NOT medic**.

QS4 Refinement. The search string *"cloud computing" AND (automated OR automatic) NEAR/2 ("resource discovery" OR "resource detection")* retrieved various duplicate results from the other queries and some false positives. Since this query was mainly focused on data retrieval from cloud environments, the term *scripting* was added because, even though they are not the focus of the query, they are commonly found within the same semantic scope. Furthermore, the terms *resource discovery* and *resource detection* were replaced by *(resource OR asset OR service) NEAR/2 (discovery OR detection OR parsing OR monitoring OR retrieval)* to increase the number of relevant results. The resulting search string is *("cloud computing" OR "cloud infrastructure") AND (automated OR automatic OR scripting) AND ((resource OR asset OR service) NEAR/2 (discovery OR detection OR parsing OR monitoring OR retrieval))*.

QS4 Refinement. The search string *("cloud cost" OR "cloud computing cost") AND ("real time" OR real-time)* was leading to very few results on most of the databases, so the terms *("real time" or real-time)* were replaced by the wildcard *optimiz** resulting in the search string *("cloud cost" OR "cloud computing cost") NEAR/2 optimiz**. The search string *finops* resulted in a relatively good number of documents. However, by adding the possibility of matching the terms *("financial operations" and "cloud computing")*, we extend the range of results while keeping the desired relevance. This results in the search string *finops OR ("financial operations" AND "cloud computing")*

Upon the completion of the refinement process, the search strings were updated to the following:

Table 3.2: Refined Search Strings

Identifier	Search String
QS1	"cloud computing" AND (real-time OR "real time") AND ("cost optimization" OR "cost management")
QS2	"cloud computing" AND cost AND (automated OR policy OR alert) AND ("recommendation engine" OR "recommendation system" OR decision-support OR "optimization system") NOT medic*
QS3	("cloud computing" OR "cloud infrastructure") AND (automated OR automatic OR scripting) AND ((resource OR asset OR service) NEAR/2 (discovery OR detection OR parsing OR monitoring OR retrieval))
QS4	("cloud cost" OR "cloud computing cost") NEAR/2 optimiz*
QS5	finops OR ("financial operations" AND "cloud computing")

3.2.3 Results

By querying the previously defined databases with the search strings defined in Table 3.2 on the databases — upon translating it to each database’s specific syntax — a total of 1003 documents were retrieved (shown in Table 3.3).

Table 3.3: Search String Results for Each Database

	Web of Science	IEEE Xplore	Scopus	ScienceDirect	Total
QS1	27	39	48	17	131
QS2	19	55	7	23	104
QS3	111	328	179	0	618
QS4	13	27	41	31	112
QS5	5	11	14	8	38
Total	175	460	289	79	1003

3.2.4 Search Filters

After gathering those initial values, a filter was applied to exclude papers published before 2018. This filter was used to ensure the results were up-to-date and relevant to the current state of the art. By applying this filter, 470 results were excluded, leaving a total of 533 documents. These documents were then exported into *Biblatex* format and imported into *Zotero* to support the following phases.

3.2.5 Duplicate Removal

Upon completing the previous steps, we leveraged *Zotero*'s duplicate detection feature to identify and remove duplicate papers. This process resulted in the removal of 140 duplicate papers, leaving 393 unique documents.

3.3 Screening Phase

The Identification Phase resulted in 393 unique documents. These documents were then submitted to Inclusion and Exclusion Criteria to ensure their relevance to this research's objectives. This criteria is displayed in Table 3.4.

3.3.1 Eligibility Criteria

The Identification phase resulted in 1003 documents, which will now be filtered based on the thirteen steps defined in Table 3.4. These steps narrow down which documents should or shouldn't be considered acceptable in this literature review.

Table 3.4: Inclusion and Exclusion Criteria for Literature Selection

Identifier	Criteria
Inclusion Criteria	
IC1	Results originate from search engines or databases.
IC2	The publication is from 2018 onward.
IC3	The publication is written in English.
IC4	The publication's content is relevant to cloud cost management or FinOps practices.
IC5	The publication focuses on cost optimization, resource management, or financial operations in cloud environments.
IC6	The publication type is a journal article, conference paper, or book.
Exclusion Criteria	
EC1	The publication is unrelated to cloud computing (e.g., medicine, agriculture, environment, etc.).
EC2	The publication is a duplicate.
EC3	The publication is a seminar or tutorial summary.
EC4	The publication discusses tangential themes or overly specific subareas (e.g., edge computing, scientific workflow scheduling).
EC5	The publication has an overly abstract focus.
EC6	The publication mentions solutions with no evidence of implementation.
EC7	The publication's full text is inaccessible.

By reading the abstract, keywords, and title, 279 documents that did not meet the criteria defined in Table 3.4 were filtered. This process resulted in 114 papers that must be submitted for a full-text analysis.

3.3.2 Full-Text Analysis

In this phase, the full text of the 114 documents considered possibly relevant to this research in Section 3.3.1 was analyzed. Nineteen documents were automatically excluded due to the inability to access the full text. After analyzing the full text of the remaining 98 papers, 76 were excluded due to the following reasons: (1) 37 documents were removed because they focused on tangential themes to **FinOps** and cloud cost management, failing to bring relevant information to this research, (2) 14 documents were removed because they were overly abstract and (3) 25 documents were removed because they were unrelated to the field of **FinOps** or cloud cost management.

3.4 Results and Discussion

The previous phases resulted in a total of 22 documents that were considered relevant to this research. This selection can be seen in Appendix A, where the selected studies, their research purpose, and the approaches or domains they focus on are presented.

Fig. 3.2 shows the number of papers in this final selection grouped by year. Observing this plot, it is evident that the number of papers published on this topic has been increasing over the last three years, indicating not only a growing interest in **FinOps** and cloud cost management but also a growing maturity in this field. This reflects the increasing focus on addressing challenges such as real-time cost optimization, resource management, and improving financial accountability in cloud environments. The growing number of published papers provides a solid foundation for identifying research gaps and establishing the context for this dissertation.

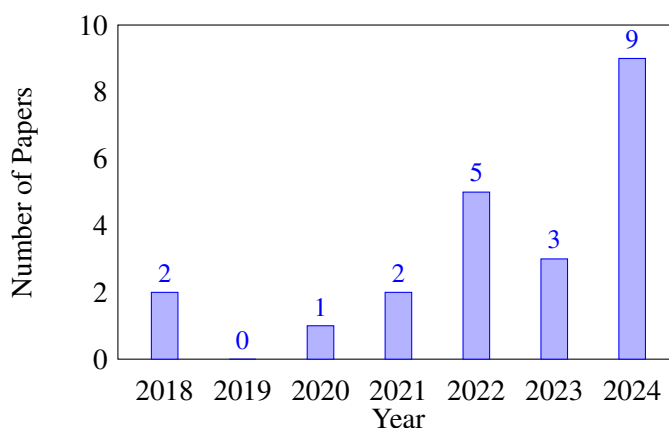


Figure 3.2: Number of papers published per year

Furthermore, the literature review conducted in the previous sections revealed that this field is still in its early stages and that there is a considerable lack of comprehensive studies that showcase the current state-of-the-art research in this field. Furthermore, we found that much of the existing literature focuses on specific environments within the vast cloud services ecosystem (e.g., computing instances, serverless computing, etc.) without a proper view of the whole picture. The following chapters aim to fill this gap by providing a synthesis of various studies and articles that were identified in the Literature Review.

3.5 Research Developments

From the literature review conducted in the previous Sections, several studies were identified that contribute to the fields of **FinOps** and Cloud Cost Management. This section provides an overview of these studies, highlighting their approaches and main contributions.

Ding et al. (2018) [32] presented a learning-based system that addresses the challenges of managing cloud costs in real time on computing instances, namely on Amazon **EC2** instances. The authors also proposed four relevant metrics — resource utilization, instance utilization, cost efficiency, and cost-saving efficiency — to more accurately evaluate the performance and cost savings of the system. By applying temporal learning algorithms, the system is also able to predict future resource demands and detect anomalies, enabling proactive cost management.

Biran et al. (2018) [13] proposed a framework to control and manage heterogeneous resources in cloud environments. This framework employs Empirical Cumulative Distribution Function (**ECDF**) to forecast demand and formulates the optimal resource allocation problem as a Mixed-Integer Linear Programming (**MILP**) problem. Upon analyzing the results, this framework proved to achieve significant cost savings compared to existing solutions. Despite the promising results, this approach may be limited by the complexity of the **MILP** problem, which may not scale well with the size of the cloud environment.

Sheikh et al. (2020) [73] explored various approaches of automation in cloud environments on the **AWS** cloud. The authors give a comprehensive overview of services such as AWS Lambda, Cloudwatch, and others and how they can be used to automate resource management tasks. Despite the focus on automation, the authors also mention techniques that can be used to optimize costs, such as automatic tagging, continuous monitoring, and weekly intimidation, to ensure that resources are being used and paid only when necessary. This approach is interesting, however it still lacks various **FinOps** practices and many of the techniques presented are not as advanced as some of the other solutions presented in this chapter.

Entrialgo et al. (2021) [33] proposed an approach to optimize cloud costs and analyze performance for transactional applications in hybrid cloud environments. They introduced *Malloovia*¹, a tool designed to optimize the allocation of virtual machines while minimizing costs and ensuring quality of service (QoS) requirements. Additionally, the authors developed *Simloovia*², a

¹<https://github.com/asi-uniovi/malloovia/tree/master>

²<https://github.com/asi-uniovi/simlloovia>

simulation tool to analyze system performance under various workloads and configurations. This study demonstrates that integrating cost optimization with simulation tools enables organizations to make more informed decisions and effectively manage their cloud expenditures.

Díaz et al. (2021) [31] conducted an analysis of the influence of billing time slot lengths — one hour, one minute, and one second — on the cost of Virtual Machine (VM) allocation strategies on the cloud. The study presented a new billing model that charges users based on the actual time they use the VMs, calculated on a per-second basis. This model, however, has a minimum charge of one minute, which means that users are charged for at least one minute of usage, even if they use it for less time. The results showed that while the per-second billing model would theoretically reduce costs, its high complexity is a barrier. Furthermore, it was found that using a one-minute time slot provides the best balance between cost and complexity, allowing users to save up to 17% compared to the one-hour time slot.

Chung (2022) [19] discussed the implementation of various FinOps practices on the AWS ecosystem. Despite the low focus on automation and on more advanced custom algorithms to identify and eliminate cloud waste as in other studies, the author thoroughly explains common FinOps practices and concepts in a more practical way, leveraging services from within AWS.

Li et al. (2022) [55] proposed a platform called SmartCMP to address challenges faced by enterprises in managing cloud costs. SmartCMP integrates various FinOps practices, enabling enterprises to continuously analyze their cloud costs in real-time, reduce waste, and speed up the decision-making process. Despite the promising claims, this research lacks comparison with other solutions and evidence of its practical application and feasibility.

Everman et al. (2022) [34] conducted a comprehensive analysis of cloud waste in Microsoft Azure. This study highlights the problems associated with having underutilized or over-provisioned resources. By introducing metrics such as Cloud Waste Points (CWPs) and Cloud Usage Score (CUS), the authors categorize and quantify the efficiency of cloud resources. Lastly, the authors propose an algorithm to identify red VMs — VMs that are underutilized or over-provisioned, leading to wasted resources and increased costs — and recommend the best allocation option maintaining Quality of Service (QoS)

Kadiyala et al. (2022) [48] introduced Kuber, a microservice deployment planner for cloud environments that optimizes deployment configurations for cost efficiency. Kuber utilizes a sample-based approach to explore all the combinations of microservices and VM types, reducing the number of runtime experiments required to identify optimal configurations. The paper states that Kuber outperformed baseline methods, identifying deployment configurations faster and with lower computational costs. However, the solution has limitations, including the need to re-run Kuber during autoscaling events and its current focus on fixed workloads.

Song et al. (2022) [80] proposed an online algorithm to optimize cost-efficient release of on-demand cloud instances in IaaS environments. This algorithm addresses challenges faced by SaaS providers, such as the management of resources under dynamic workloads, allowing for real-time decision-making without knowledge of future requirements. The study highlights the algorithm's low computational overhead and ease of implementation, making it a practical solution. Despite

giving valuable insights, the study focuses more on the point of view of a **SaaS** provider and lacks details on how the algorithm could be implemented on a service-level basis.

Khan et al. (2023) [50] presented a novel approach to cloud cost management by proposing a graph-based model for representing cloud resources, making it easier to apply optimization techniques. The authors use graph theory, mentioning a six-path optimization process that includes strategies for network and storage cost optimization, and resource placement. Their findings suggest that the proposed model can minimize costs and improve performance in cloud environments.

Yehoshua et al. (2023) [91] presented *CCO (Cloud Cost Optimizer)*, a cloud cost optimization framework that leverages meta-heuristics such as Tabu Search and Simulated Annealing to find optimal deployment configurations for cloud workloads. Despite the relatively short size of the paper, CCO is open-source on GitHub [25] and, after a brief analysis, seems to serve as a good baseline for further development. Having said this, some limitations were identified, such as the specialization of the framework for computing instances and the lack of maintenance and updates during the last year.

Chhetri et al. (2023) [18] proposed a proactive risk-aware cloud cost optimization framework that utilizes low-cost resources, such as Spot instances, to reduce expenses while considering the risks of losing these resources. They introduce two key strategies: (1) Contract Diversification (CD) combines low-cost resources with more stable options to reduce the risk of revocation. (2) Resource Diversification (RD) spreads the risk across different types of low-cost resources. This framework aims to optimize the resource portfolio for long-running applications by balancing cost savings with acceptable levels of revocation risk, measured using metrics like Maximum Revocable Capacity (MRC) and Total Cost Savings (TCS). Their simulations, which used real workload data and Amazon **EC2**, show that this method can significantly reduce costs while keeping the risk of resource loss within reasonable limits.

Murugesan (2024) [62] discussed various cost optimization techniques on **AWS**, such as right-sizing resources and making use of different pricing models offered. Furthermore, the author also mentioned the importance of continuously monitoring and tagging resources for effectively managing cloud costs. This paper provides a good overview of simple cost optimization techniques that can be applied in **AWS** while giving a good overview of best practices.

Naarayanamy et al. (2024) [63] proposed an AI-powered **FinOps** strategy to optimize cloud computing costs by collecting usage patterns, predicting future demands, and recommending measures to reduce costs. The authors set a sound theoretical base for further research; however, as far as this research could find, the proposed solution or results were not made available by the authors for testing or further development.

Shende and Chandak (2024) [74] presented a framework to optimize cloud costs by integrating automated analytics and resource allocation strategies. The author's approach leverages real-time data collection from major **CSPs**, predictive analytics for forecasting demand. By integrating Infrastructure as Code (**IaC**) tools like Terraform and Ansible for automated management, this theoretical framework aims to increase operational efficiency and reduce costs.

Srivastava et al. (2024) [83] presented cloud service monitoring techniques focused on optimizing performance and cloud cost management. They propose three strategies: (1) real-time performance monitoring to track key metrics, (2) predictive analytics for anomaly detection and (3) cost optimization through resource scaling. These approaches enhance organizations' efficiency and financial performance in cloud environments.

Ponnusamy and Khoje (2024) [66] explored how to use Machine Learning (ML) techniques to optimize cloud costs through predictive resource scaling strategies. This study also covers new trends such as explainable Artificial Intelligence (AI), ML, and their ethics. The authors conclude that these strategies are essential for businesses to succeed in the competitive cloud computing market. Despite the interesting insights, this study lacks practical application and serves more as a theoretical overview of the topic.

Smendowski and Nawrocki (2024) explored multi-time series forecasting techniques, such as the development of a multi-time series forecasting system (MSFS) and a similarity-based time-series grouping (STG) method. They also introduce a hybrid ensemble algorithm for anomaly detection (HEADA), adding to the system's predictive capabilities. Their research highlights the importance of accurately forecasting resource utilization in dynamic cloud environments, as workload variations can lead to cost spikes if not effectively managed. By integrating these methodologies, the authors demonstrate how organizations can optimize resource allocation, reduce operational expenses, and improve overall QoS in cloud environments.

Roh et al. (2024) [70] proposed an efficient switching mechanism between VMs and serverless computing that leverages a Proportional-Integral-Derivative (PID) controller to easily switch between the two paradigms based on metrics such as traffic load, cost, and execution time. Their results demonstrated measurable cost savings compared to traditional approaches.

Biswas and Kumar (2024) [14] explored serverless computing but from a different perspective. They developed a dynamic adaptive scaling model for serverless computing to enhance resource management through real-time monitoring and predictive analytics, leveraging algorithms such as Random Forest and Q-Learning. The paper concludes that the proposed model significantly improves operational efficiency and reduces costs, achieving an improvement of up to 30% in resource utilization and a reduction of 25% in operational expenses. It also remarks on the possible application of more complex machine learning algorithms to further improve the model's performance.

Yang et al. (2024) [90] proposed a FinOps framework to optimize IT FinOps and reduce the carbon footprint of cloud computing through unsupervised workload characterization. The general idea of this framework is to classify workloads as two distinct types — active and inactive. When repeated enough, this characterization allows the framework to label a workload as productive or non-productive and, therefore, de-provision unused resources. This paper provides some architectural insights on how the framework could be implemented and some theoretical formulas; however, there is still some complexity involved in implementing this framework.

3.6 Analysis of Research Developments

After the analysis of the papers conducted in Section 3.5, it is possible to label each paper according to its characteristics to achieve a better understanding of the current state of the art. For that purpose, the following characteristics were defined:

Service. Some papers focus on specific services within a cloud provider. The same service across different providers may have some differences, but for this analysis, we consider them the same. Examples of services are Computing Instances (Virtual Machines), Serverless Computing, and Kubernetes.

Cloud Type. According to National Institute of Standards and Technology (NIST) [59] there are four cloud deployment models — Private, Public, Hybrid, and Community Cloud. For the following analysis, we considered Multi-Cloud as a separate category, as it provides a different set of challenges and possibilities compared to more specific solutions.

Focus. The majority of the selected papers focus on **FinOps** and cost optimization; however, some might be more specific or tangential. For that reason, Focus is a broader term that encompasses the paper's main goal. Possible focuses are Cost Optimization, Resource Allocation, Waste Reduction, and Risk Management.

Key Technique. Various techniques are used to achieve the goals presented in each paper, often combining multiple methods. The Key Technique summarizes the main approach used by the authors to solve the problem. When various are used, the most relevant is chosen. Examples of techniques are Machine Learning, Mixed-Integer Linear Programming, and Q-learning.

Provider. The cloud provider is addressed in the paper. While some papers address multiple providers or are provider-agnostic, others are more specific. Examples of providers are Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP).

License. A software license is essential to know to which extent we can use and modify the code provided by the authors. Possible licenses are MIT, GPL v3, and Apache 2.0.

Artifacts. An artifact refers to any material that can be used to replicate the study. This includes source code, formulas, in-depth descriptions, and any other format, as long as it is transparent and detailed. The artifact field is classified as follows: (1) results that provide enough artifacts to replicate the studies completely, (2) results that provide some artifacts but not enough to replicate the study fully, and (3) results that provide little to no artifacts.

Table 3.5: SoTA literature and their characteristics. hyphen (-) means no information available, checkmarks (✓) means yes, dots (•) means incomplete, empty means no, and asterisk (*) means various.

Reference	Service ¹	Cloud Type ²	Focus	Key Technique	Provider	License	Artifacts
Biran et al. [13]	VM	P	Resource Allocation	MILP	*	-	•
Ding et al. [32]	VM	P	Cost Monitoring	Temporal Learning	AWS	-	•
Sheikh et al. [73]	VM ³	P	Automation	IaC	AWS	-	•
Entrialgo et al. [33]	VM	H	VM Optimization	Simulation	*	MIT	✓
Díaz et al. [31]	VM	P	VM Allocation	Per-Second Billing	*	-	•
Chung [19] ⁴	*	P	FinOps Practices	-	AWS	-	✓
Everman et al. [34]	VM	P	Waste Reduction	Metrics Analysis	Azure	-	-
Kadiyala et al. [48]	VM	P	Resource Allocation	Constraint Programming ⁵	AWS	-	✓
Li et al. [55]	-	P	Cost Optimization	-	*	-	-
Song et al. [80]	-	P	Instance Release	Online ML	*	-	•
Chhetri et al. [18]	VM	P	Risk Management	Portfolio Diversification	AWS	-	•
Khan et al. [50]	*	M	Cost Modeling & Optimization	Graph Theory	*	-	-
Yehoshua et al. [91]	VM	P/H	Cost Optimization	Meta-Heuristics	AWS/Azure	GPL v3	✓
Biswas and Kumar [14]	SRV	P	Resource Auto Scaling	Q-Learning	*	-	•
Murugesan [62]	*	P	Cost Optimization	*	AWS	-	•
Naarayanamy et al. [63]	-	-	Cost Optimization	ML	-	-	-
Ponnusamy and Khoje [66]	-	P	Cost Optimization	ML	AWS	-	-
Roh et al. [70]	VM, SRV	P	Cost Optimization	PID Controller	AWS ⁶	-	-
Shende and Chandak [74]	VM	P	Cost Optimization	IaC, ML	* ⁷	-	-
Smendowski and Nawrocki [79]	VM	P	Resource Utilization Maximization	Time Series Forecasting	GCP ⁸	-	-
Srivastava et al. [83]	-	-	Cloud Monitoring	Real-Time Monitoring	-	-	•
Yang et al. [90]	VM	P/H	Workload Optimization	Unsupervised Learning	-	-	-

¹ Virtual Machines (VM), Serverless Computing (SRV).

² Public Cloud (P), Hybrid Cloud (H), Multi-Cloud (M).

³ Despite the focus on VMs (EC2), the techniques are easily adaptable to other services.

⁴ This book focuses on various FinOps practices across multiple AWS services.

⁵ No evidence of using constraint programming is present in the paper; however, after further analyzing the source code, it is clear that the authors use this technique.

⁶ The authors use AWS services for their experiments; however, the proposed mechanism seems adaptable to other cloud providers.

⁷ Despite the lack of practical evidence in the paper, the authors claim this approach can be used in AWS, Azure, and GCP environments.

⁸ No mention of the cloud provider is made by the authors; however, by observing the tests, it is easy to infer the environment was Google Cloud.

By analyzing Table 3.5, we can observe that most papers focus on computing instances — also known as Virtual Machines (VMs). This is expected, as VMs are the most common service used by organizations and the first to be created.

Public cloud is the most common cloud type addressed in this paper, which reflects the current market trends. Various organizations choose this cloud model due to its ease of use, scalability, and wide range of services.

Machine Learning is the most used technique across this papers, showcasing the growing importance of this technology in the field of FinOps, enabling organizations to predict future demands, optimize resource allocation, and reduce costs based on historical data.

Moreover, it is possible to observe that few articles provide enough artifacts to replicate the study straightforwardly. This research showed that this seems to be a common issue in the field of FinOps, Cloud Cost Management, and tangential areas. Furthermore, some of the solutions presented in the literature were found in the form of closed-source tools or paid services. This limits the ability of researchers to test and extend these solutions and may delay the development of new research in this field.

3.7 Industry Trends and Market Analysis

The rapid growth of cloud computing in the last decade has increased the complexity of cloud environments. This complexity is also increased by the wide range of services and pricing models offered by **CSPs**. As a result, cloud tenants' awareness of their cloud costs is becoming increasingly important. This is especially true for organizations looking to optimize their cloud costs and maximize their Return on Investment (**ROI**). Therefore, a market analysis is presented in this section to provide a clearer view of the current trends, existing solutions, limitations, and opportunities for growth in this field.

The **FinOps** and Cloud Cost Management market are growing more than ever, with companies increasingly relying on cloud services to run their businesses. Among the various trends that are shaping the market, the following were found to be among the most prominent:

Organizations are increasingly adopting **FinOps** practices to reduce cloud costs, driven by initiatives such as the FinOps Foundation [39]. According to the State of FinOps Survey 2024 [87], more than half of the organizations stated that Automation was their top priority for the following year. This trend is also reflected by the number of tools and services that are being developed to automate cloud cost management tasks.

The adoption of **AI** and **ML** is also being increasingly adopted in this domain. In recent years, these technologies are starting to be experimented within the context of cloud cost management, with 46% of organizations who answered the State of FinOps Survey 2024 [87] stating that they are using or planning to make usage of these techniques in their cloud cost management practices in a near future.

Additionally, as the environmental impact of cloud computing becomes more and more evident [61], various organizations are shifting towards more sustainable practices and looking for ways to reduce their carbon footprint. Although this is not a new trend, it may influence future cloud cost management practices, with major **CSPs** such as **AWS** setting ambitious goals to become carbon neutral by 2040 [86].

3.7.1 Existing Solutions

Managing cloud costs effectively has become a critical challenge for organizations as they must deal with increasingly complex and scalable cloud environments. To address this, a range of tools and services has been developed. These can be broadly categorized into native tools provided by **CSPs** and third-party solutions created by independent companies.

Major **CSPs** such as **AWS**, **GCP**, and Azure provide native tools to help their customers manage their cloud costs, offering dashboards and cost advising services [20, 22, 60]. These tools are a good starting point for tenants to understand and improve their cloud costs. Furthermore, they provide accurate metrics and are integrated within the respective **CSP** ecosystem, making them easier to use and understand. However, these tools are often limited in terms of customization and optimization. Furthermore, these solutions may cause a delay in reporting the costs, making it difficult for tenants to react proactively to cost spikes before they happen. For instance, AWS Cost

Explorer provides a comprehensive view of the costs, but it may take up to 24 hours to update the cost data [15, 72].

To overcome the limitations of these native tools and provide some more advanced features, several third-party **FinOps** tools have been developed — some open-source, others sold as a service — to help organizations manage their cloud costs more effectively. Among these tools, the ones that stood out in this research for their relevance to this dissertation’s goal, features, popularity, and ability to verify their claims are presented below:

CloudZero [27] is a cloud cost intelligence platform that empowers organizations to understand and optimize their cloud costs. It provides real-time insights into cloud costs, usage, performance, and other metrics. Furthermore, it has integrations with various **CSPs** such as **AWS**, **GCP**, and **Azure**. It provides data normalization across all of these. *Cloudzero* provides a cost allocation feature that allows organizations to allocate costs to different departments, teams, or projects. It also leverages **AI** for automatic anomaly detection and many other features. Overall, *CloudZero* seems to be a very comprehensive solution for cloud cost management, however it is not open-source and may be a bit too expensive for Small and Medium-sized Enterprises (**SMEs**).

Infracost [75] is an open-source tool that allows developers to estimate the cost of their infrastructure before deploying it. It parses Terraform and provides detailed cost estimates for each resource in the template. Although specialized, *Infracost* appears to be a promising tool with potential for various Continuous Integration/Continuous Deployment (**CI/CD**) integrations.

Opencost [64] is an open-source tool that provides real-time cost monitoring and cost optimization insights within the Kubernetes ecosystem. It gives organizations a more transparent and unified view of their cloud costs, provides insights and recommendations for cost optimization, and offers the option to set customizable real-time alerts for cost spikes, anomalies, underutilized resources, and more.

Spot.io [17] leverages advanced analytics and **ML** algorithms to continuously optimize cloud environments for cost, performance, and availability. It gives tenants more visibility and control over their cloud costs, automatically provisioning and scaling their cloud resources based on a real-time workload analysis.

nOps [24] is an autonomous cost optimization platform for **AWS** that, similarly to the other tools, enables visibility over cloud costs and helps organizations to get the most out of their cloud investments and also be able to scale their resources automatically based on usage. *nOps* offers a wide variety of **AWS** cost optimization features for different services such as **EC2**, Relational Database Service (**RDS**), **EKS**, and others. Like some of the previously mentioned tools, *nOps* is also closed-source and, as far as this research could find, does not provide pricing information on its website.

finout [37] is a **FinOps** platform for enterprises and, similarly to *CloudZero*, it leverages various features — such as cost allocation, virtual tags, and automatic connection with BI tools — that are not central to this research. However, *finout* also provides cost optimization tools such as automatic scaling, rightsizing, automatic shutdown policies, and continuous scans on the cloud

environment to identify cost-saving opportunities. *finout* is also closed-source, and some of the features are limited to a subset of services, depending on the CSP.

3.7.2 Analysis of Market Solutions

The tools presented in Section 3.7.1 present a wide range of features and capabilities that can empower organizations looking to optimize their cloud costs. However, they also present some limitations that should be considered when adopting them.

Native tools provided by CSPs have the advantage of being integrated within the respective cloud ecosystem, providing accurate metrics and data in a user-friendly and familiar interface. However, they are often limited in terms of customization and granularity of the data and, as expected, lack some of the more advanced features provided by specialized third-party tools. Furthermore, these native tools may delay reporting the costs, making it more difficult for tenants to react proactively to cost anomalies before they happen [15, 72].

Third-party tools, on the other hand, provide a more comprehensive or specialized set of features that can help organizations optimize their cloud costs more effectively and circumvent some of the limitations of the native tools. To correctly characterize the tools presented in Section 3.7.1, it is essential to define some characteristics apart from the ones presented in Section 3.6. These characteristics are:

Open Source. Open-source tools allow users to inspect, modify, and adapt them. Furthermore, in this context, open-source tools allow for options such as self-hosting.

Access. The access to a product may be done through various means, some more straightforward than others. Access types include requesting a demo via a form or email, signing up for an account, contacting sales for more information, or hosting on-premises.

Pricing. The pricing model is an essential feature in weighing the cost-benefit ratio of a product before acquiring it. For that, pricing models should be as transparent as possible. Possible pricing models are free-tier, custom pricing plans based on the user's needs, fixed pricing schemes, and pay-as-you-go models.

Table 3.6: Industry solutions and their characteristics. Hyphen (-) means no information available, checkmarks (✓) mean yes, N/A means not applicable, and empty cells represent no.

Tool	Service	Cloud Type ¹	Provider	License	Open Source	Access	Pricing ²
CloudZero	*	M	*	-		Request Demo, Sign Up	-
Infracost ³	N/A	N/A	N/A	Apache 2.0	✓	Self-Hosted, Sign Up	F, C, X
Opencost	Kubernetes	M, H	*	Apache 2.0	✓	Self-Hosted, Contact Sales	F, C
Spot.io	- ⁴	M	*	-		Request Demo, Sign Up	-
nOps	*	P	AWS	-		Demo, Sign Up	-
Finout	*	M	*	-		Sign Up	X, C, P

¹ Public Cloud (P), Hybrid Cloud (H), Multi-Cloud (M).

² Free-Tier (F), Custom (C), Fixed (X), Pay-as-you-go (P).

³ Infracost focuses on infrastructure by parsing Terraform templates to provide cost estimates for each resource, so it is not tied to a service, cloud type, or cloud provider.

⁴ Spot.io seems to offer a wide range of service support; however, as far as this analysis goes, it is unclear which services are supported.

As shown in Table 3.6, tools often incur additional fees and may not be accessible to all organizations, especially **SMEs**. Some of these tools also lack transparency in their pricing and usability, being accessible through a sales form, making it more difficult for potential customers to make an informed decision. Furthermore, most of the tools discovered in this analysis are closed-source, which may limit the ability of organizations to further customize and extend them to their specific needs.

3.8 Summary

This chapter presented the current State-of-the-Art in **FinOps** and Cloud Cost Management from an academic and industry perspective. Academically, we can observe a trend towards developing more advanced algorithms and models to optimize cloud costs, leveraging techniques such as Machine Learning, Reinforcement Learning, Mixed-Integer Linear Programming, and others. On the industry side, the ecosystem of **FinOps** seems to be steadily growing, with an increasing number of tools and services being developed aiming to empower organizations with a more transparent view of their cloud costs and helping them to optimize the usage of their cloud resources automatically or semi-automatically. Furthermore, there seems to be a growing concern to address cloud costs and normalize **FinOps** practices, with the arrival of organizations such as Finops Foundation [39].

Despite the advancements in this field, various gaps and opportunities for growth can be identified. Academic solutions often focus on specific aspects of cloud computing or a particular subset of services without providing a comprehensive view of the whole cloud ecosystem. Furthermore, various studies presented in Section 3.5 are limited to theoretical models and lack enough evidence of their practical application, as shown in Table 3.5. On the other hand, various of the industry solutions presented in Section 3.7 are mostly closed-source and lack transparency in their pricing, usability, and features, making it difficult to prove their claims and evaluate their effectiveness without a proper trial.

These findings showcase the relevance of this research and the importance of developing a comprehensive, transparent solution that can bridge the gap between the academic and industry perspectives.

In the following chapters, a novel **FinOps** framework will be proposed, leveraging some of the latest advancements in this field and aiming to provide a more transparent and effective solution for cloud cost management.

Chapter 4

Problem Statement and Methodology

This chapter builds upon the previous findings and aims to explore the work conducted during this dissertation in greater depth. It also explains how the research was conducted, detailing how the research questions were addressed and the design principles that guided the development phase of this dissertation.

Section 4.1 outlines this dissertation’s problem statement and motivation. Section 4.2 presents the central hypothesis that this dissertation aims to validate or falsify. Section 4.3 describes the approach taken to answer the research questions. Section 4.4 outlines the major design principles that guided the development of the FinOps framework, providing a high-level overview of its architecture, and discussing its scope, limitations, and expected outcomes, including the services that will be supported. Section 4.5 discusses the expected outcomes of this research. Finally, Section 4.6 introduces the user demand evaluation that will be conducted to gather feedback from potential users of the work developed in this dissertation.

4.1	Problem Statement	32
4.2	Hypothesis	33
4.3	Research Questions	33
4.4	Solution Overview	35
4.5	Expected Outcomes	36
4.6	User Demand Survey	37

4.1 Problem Statement

As outlined in Section 1.2, managing cloud costs is essential to ensure sustainable and efficient cloud usage. However, despite the existence of native cloud cost management tools, such as AWS Cost Explorer and Amazon Cost and Usage Reports (CUR), these reports might appear with a delay of up to 24 hours [15, 72], making it possible to incur unwanted costs due to the lack of real-time knowledge.

On top of that, these cost reports are often not as transparent and granular as desired, requiring significant manual effort — for example, by downloading and analyzing CSV files or setting tags on resources to track costs — and some financial expertise to interpret them. This means that tenants need to spend additional time manually analyzing these reports or even hiring financial experts to interpret them, which may impact financial management and distract them from their core activities, leading to an increased operational burden.

Additionally, pricing models depend highly on the **CSPs** and service-specific characteristics [58]. This adds a layer of complexity, making it even more challenging to estimate, compare, and optimize cloud expenses. Even with the proper expertise, this process is very prone to error.

4.2 Hypothesis

As a result of the lack of real-time visibility over cloud costs and resources, the ever increasing complexity of cloud environments, and the lack of transparency and granularity in existing cloud cost management tools, tenants may struggle to manage their cloud costs and resources effectively, even with expertise in cloud computing and financial management. In some cases, this can lead to unexpected costs, inefficiencies in resource allocation, and a lack of control over cloud expenses, which can ultimately impact the financial health of the organization.

Considering these problems, the central hypothesis to this dissertation was formulated:

H: *A system that monitors and forecasts costs in real-time can provide decision-makers with indicators that lead to measurable cost savings and better resource allocation.*

A system that monitors and forecasts costs in real time refers to a system that can gather real-time data from a cloud ecosystem and predict future costs based on historical usage data.

Decision-makers refer to cloud administrators or tenants who have the authorization to manage cloud resources and access to cost data.

Indicators that lead to measurable cost savings and better resource allocation imply that the cloud costs using this system are less — assuming a non-optimized or imperfect setup — than the costs using previous cost management solutions on an equivalent time frame after the recommendations made by the system are applied.

4.3 Research Questions

With the widespread adoption of cloud computing, new cloud-native services, pricing models, and cost management tools have emerged. However, as outlined in Section 4.1, this growth of choices has also led to increased complexity in managing cloud costs; and the existing native cost management tools often do not provide real-time insights, which can lead to unexpected costs and inefficiencies.

To address these challenges and contribute to the current body of knowledge, we consider the fundamental desideratum of this dissertation **to develop a system that enables tenants to monitor, analyze, and optimize their cloud costs and resources in real time**, while also providing actionable recommendations to reduce these costs and improve resource allocation.

Given that, this dissertation formulates five research questions to guide this research and help answer the hypothesis presented in Section 4.2:

- RQ1: Which tools can be used to retrieve data from cloud environments in real-time?** Do they require any licensing? Do they support various **CSPs**? Do they incur additional costs?
- RQ2: Which algorithms and heuristics should be developed to catalog cloud resources, monitor costs, and optimize resource allocation?** Are there other metrics that should be taken into account? Do they need historical cloud cost data?
- RQ3: Which actions or policies can be recommended to reduce cloud costs, improve resource efficiency, and enable more informed decision-making?** How can policies be tailored to different service cost models? How could forecasting based on previous usage affect this? To what extent can this be automated?
- RQ4: How do real-time cost monitoring and actionable recommendations improve a tenant's financial operations compared to cost reports?** What are the benefits regarding cost savings and operational expenses?
- RQ5: How can the system be integrated into existing environments?** What are the requirements for integration? How can it be made compatible with existing tools and workflows?

The approach to answering these questions was divided into three logical phases: (1) Research phase, which includes the literature review and State-of-The-Art analysis; (2) Development phase, which includes the design and implementation of the solution; and (3) Validation and evaluation phase, which includes the testing and validation of the solution.

The research phase focuses on gathering enough information to understand the problem domain, which solutions are already available, and what gaps exist. During this phase, we gathered some insights about the tools and techniques used both in the industry and in the academia to address problems such as resource discovery, cost optimization, and real-time observability, thus allowing us to be closer to answering **RQ1** and **RQ2**. The development phase focuses on developing the proposed solution, which includes designing and implementing the FinOps framework. This phase is guided by the insights gathered in the first phase and the experience gained throughout the development process. Having this said, this phase is focused on answering **RQ3** and completing the answers to **RQ1** and **RQ2**. The validation and evaluation phase is focused on testing the developed solution and validating whether it meets the requirements and expectations set in the previous phases. This phase is essential to verify or falsify the hypothesis presented

in Section 4.2 and ensure that the solution effectively reduces cloud costs and improves resource allocation, thus answering **RQ4**. Since we will use real-world data for validation, this phase will also help us answer **RQ5** by providing insights on how the system can be integrated into existing cloud environments and the requirements for integration.

4.4 Solution Overview

This section outlines the main design principles that will guide the development of the solution. These principles arose from the insights gathered during the research phase and are intended to ensure that the solution is effective, efficient, and easy to use. The principles are as follows:

Cost Optimization Algorithms: The solution will implement algorithms and heuristics to optimize cloud costs. These algorithms will be designed to fetch data from Prometheus, process it, and provide more complex insights into the cloud resources, costs, and usage. The results of these algorithms will also be stored in Prometheus, allowing for easy integration with Grafana dashboards and AlertManager alerts.

Real-Time Observability: Real-time observability will be achieved by fetching real-time metrics¹ from solutions such as Amazon CloudWatch [9] or Opencost [64], storing them in a database such as Prometheus [67], and then processing them and displaying them in Grafana dashboards. The framework will then use these metrics to provide insights into the cloud resources, costs, and usage. This will enable users of this solution to monitor their cloud environments effectively and take proactive actions to reduce costs, while avoiding the delay associated with services such as Cost Explorer [22] and Amazon CUR [15, 72].

Alerting and Notification Mechanisms: For the alerting and notification mechanisms, the solution uses AlertManager, which is a component within the Prometheus ecosystem. Alerting rules are defined in Prometheus, and use AlertManager to notify users of potential cost spikes or anomalies, as well as to suggest actions for cost reduction.

Extensibility: The solution should be designed to allow users to easily extend it with new algorithms and heuristics — either by adding new modules for new services or by modifying existing ones. This will enable more experienced users to adapt the framework to their specific needs and use cases, while allowing for future enhancements in existing algorithms and heuristics. For that, the codebase should be modular, well-documented, and follow best practices for software development, such as using adequate design patterns. It also should opt for open-source libraries and tools whenever possible.

User Experience: The solution should be easy to use and integrate with Kuehne + Nagel infrastructure. For this, the solution will be designed to work with existing open-source tools

¹Real-time metrics refer to metrics that are collected and processed in near real-time.

and technologies used by Kuehne + Nagel, such as Prometheus, Grafana, and Alertmanager. The solution should also provide clear documentation and examples to help users understand how to use the framework and its features and integrate them into their existing workflows. Furthermore, it should offer a user-friendly way of configuring the framework without the need for extensive knowledge of the codebase or underlying technologies. This could be achieved by providing a configuration file in YAML or JSON format.

By following these design principles, the solution aims to provide a comprehensive and effective FinOps framework that can help cloud tenants monitor, analyze, and optimize their cloud costs and resources in real-time, while also providing actionable recommendations to reduce cost inefficiencies and prevent unexpected costs. Since the solution is built with extensibility and user experience in mind, it can be easily adapted to different cloud services and environments without requiring significant changes to the codebase or deep technical expertise from the users.

From a high-level perspective, the framework could be divided into four main steps: (1) fetching various metrics such as CPU utilization, storage cost, network bandwidth, and others from Amazon CloudWatch [9] or Opencost [64] and store them in Prometheus [67] grouped by service, (2) run the respective service's algorithm or heuristic on the most recently retrieved data, and, possibly given historical data as input for some services, (3) store the results in Prometheus, (4) create dashboards with Grafana [41] to enable real-time observability of the cloud resources, costs, and usage, (5) setup alerts on Prometheus with AlertManager [4] to notify the user of possible cost spikes or anomalies and to suggest actions to be taken for cost reduction.

The codebase will be implemented in Python due to its ease of use, extensive library support, documentation, and seamless integration with Prometheus and Grafana. For the implementation of the algorithms, the codebase will be designed to allow for easy integration of new algorithms and heuristics.

Furthermore, due to the complexity of the problem domain and the vast number of cloud services available, this dissertation will focus on a subset of cloud services, which were collectively identified as the most relevant for Kuehne + Nagel's current use cases. These services are Amazon EC2, and Amazon EKS. Having this said, the framework will be designed to be extensible, allowing for easy addition of new services in the future. The following Section provides a clearer separation of these frameworks and their scopes.

4.5 Expected Outcomes

In Chapter 3, several gaps between the literature and the market were identified. The literature mainly focused on novel approaches leveraging complex algorithms applied to a narrow subset of cloud computing. In contrast, market solutions focused on a broader and sometimes less granular view of cloud cost reduction possibilities. Furthermore, details of the implementation of these solutions are challenging to find in both scenarios, either because they are closed-source or not mentioned at all.

Having this said, this dissertation expects to deliver a FinOps framework² that can be used by cloud tenants to have proactive control and real-time observability over their cloud resources, costs, and usage. While the framework is conceptually unified, it will accommodate different implementations tailored to specific cloud services. These implementations address the unique characteristics of each service and the particular requirements and limitations that arose during the research and development phases. As a result, the solution will consist of two frameworks:

CloudWatch-based Framework: This framework will be designed to work with Amazon EC2 instances, making use of Amazon CloudWatch to gather real-time data about the instances, such as CPU utilization, network bandwidth, and others.

Kubernetes-based Framework: This framework will be designed to work on top of any Kubernetes cluster — including, but not limited to Amazon EKS — and will be developed on top of opencost [64].

From a more conceptual view, this dissertation also aims to develop a solution that can reduce costs in cloud environments, compete with existing State-of-The-Art solutions, and provide a generalized set of algorithms to reduce costs in these services. Furthermore, the framework that will result from this work should be easy to use and integrate with existing infrastructures — by providing clear documentation and examples to help users understand how to use the framework, its features, and how to integrate it into their existing workflows. This dissertation will aim for future integration and compatibility with Kuehne + Nagel’s ecosystem and will be built upon open-source solutions.

The following chapters will explain these frameworks’ design, implementation, and validation in more detail.

4.6 User Demand Survey

The complete survey results, including detailed demographic breakdowns and feature evaluations, are presented in Appendix F.2.

Since the frameworks developed in this dissertation are intended to be used both by Kuehne + Nagel and the wider FinOps community, it is essential to evaluate whether the functionalities provided by these frameworks align with user needs and expectations. To validate the practical relevance of the proposed solution and understand the current state of cloud cost management practices, a comprehensive user demand survey was conducted among cloud computing professionals.

²The term "framework" is used to refer to the overall methodology and approach, rather than a single unified tool.

4.6.1 Survey Overview

The user demand survey was designed to gather insights from potential users regarding current challenges in cloud cost management and the perceived value of real-time cost monitoring solutions. The survey was distributed to 21 cloud computing professionals, including DevOps engineers (38.1%), software developers (28.6%), engineering managers (19.0%), and site reliability engineers (9.5%) from organizations of various sizes.

The survey targeted professionals with substantial cloud computing experience, with 90.5% of respondents currently working with cloud platforms and 85.7% having Kubernetes experience. Within the participants, 85.7% of respondents came from large organizations, according to the European Commission's definition of enterprise sizes [40], providing insights into enterprise-scale cloud cost management challenges.

4.6.2 Key Findings: Validating the Problem Statement

The survey results (refer to Appendix F, Section F.2) reveal a fundamental disconnection between the perceived importance of cloud cost management and the available tools and practices to achieve it. Figure 4.1 shows that while 76.2% of organizations report that cloud costs play a significant role in their decision-making processes, only 38.1% have excellent visibility into their cloud costs.

4.6.2.1 Cost Visibility Gap

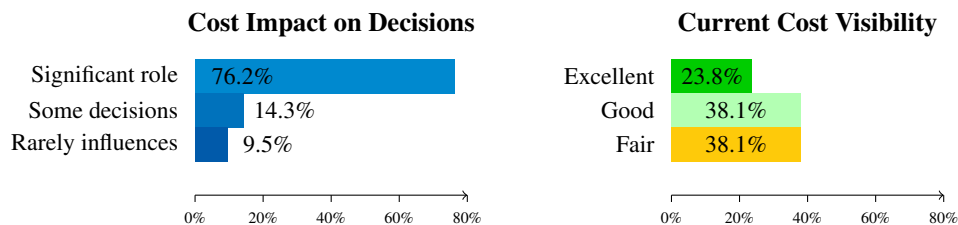


Figure 4.1: Cost impact versus visibility gap: 76.2% of organizations consider costs significant for decision-making, yet only 23.8% have excellent cost visibility, demonstrating a fundamental disconnect between the importance of cost management and available tools.

This gap validates one of the main problems of this dissertation: organizations are making business-critical decisions based on cloud cost considerations while lacking the comprehensive visibility needed to make these decisions effectively. The remaining respondents reported having good (38.1%) or fair (38.1%) visibility into their cloud costs, indicating they can see costs but lack detailed insights or have only basic cost tracking with limited analysis capabilities, respectively.

4.6.2.2 Current Cost Management Challenges

The survey identified several critical pain points that directly support the motivation for developing real-time cost monitoring frameworks, as illustrated in Figure 4.2:

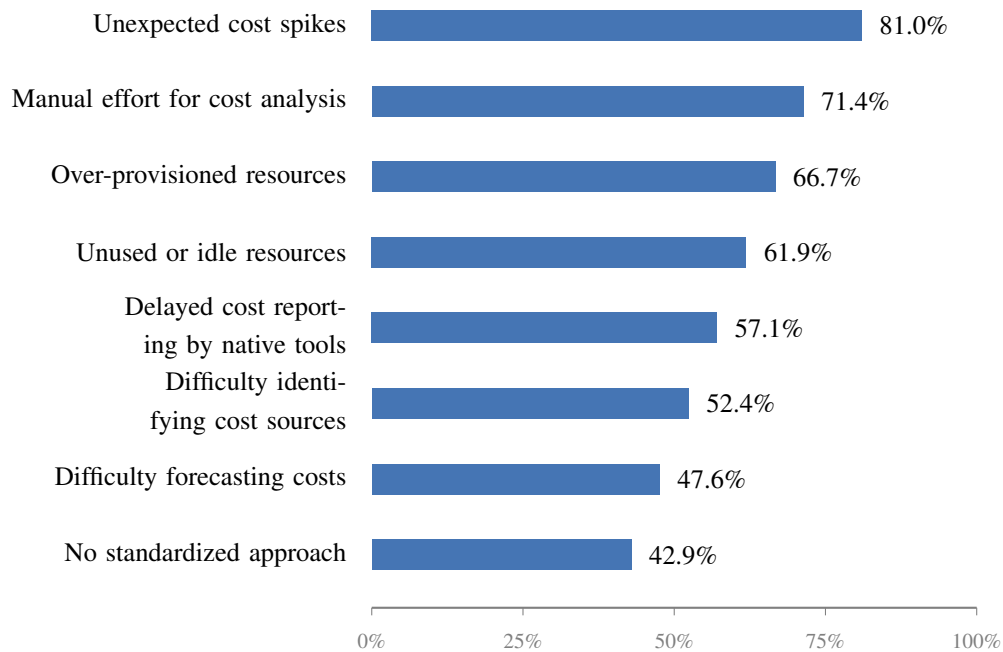


Figure 4.2: Primary cloud cost management challenges faced by surveyed organizations, revealing that unexpected cost spikes (81%) and manual analysis burden (71.4%) are the most significant operational pain points affecting cloud cost management effectiveness.

The survey results highlight several key challenges in cloud cost management. Among them, the most significant challenges identified were:

Unexpected Cost Spikes: The most significant concern, affecting 81% of organizations, highlights the reactive nature of current cost management practices. This finding directly supports the need for proactive, real-time monitoring solutions.

Manual Analysis Burden: 71.4% of respondents struggle with the manual effort required for cost analysis, with 90.5% spending time on manual processes monthly. Notably, 47.6% dedicate six or more hours per month to these activities, representing a substantial operational burden.

Resource Inefficiencies: 66.7% of respondents reported having problems with over-provisioned resources and 61.9% with unused or idle resources, indicating significant optimization opportunities that current tools fail to address effectively.

These findings showcase the need for a solution that can provide real-time visibility into cloud spending, automate cost analysis, and optimize resource allocation to address the identified challenges effectively.

4.6.3 Framework Demand and Adoption Potential

The survey results showcased a strong demand for the proposed frameworks. As illustrated in Figure 4.3, all respondents indicated positive adoption likelihood, with 72.2% being very likely

to use the frameworks and 27.8% being likely to adopt them. This universal interest represents a consensus among respondents regarding the value proposition of the frameworks.

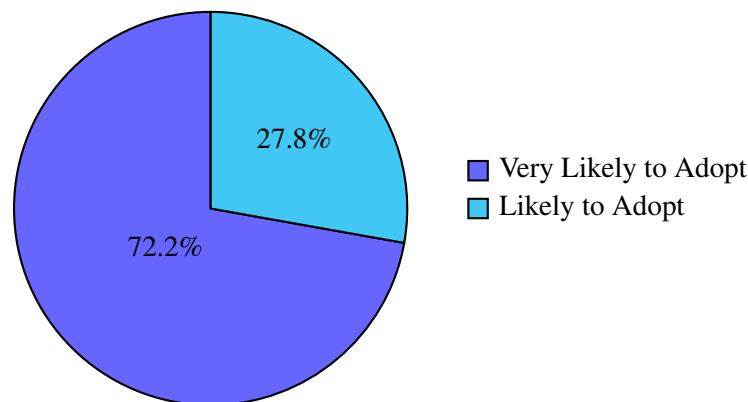


Figure 4.3: Framework adoption likelihood showing universal positive interest, with 72.2% of respondents very likely to adopt and 27.8% likely to adopt the proposed solutions.

Furthermore, Figure 4.4 illustrates the cost reduction expectations of respondents, showcasing that 72.2% of respondents would implement the frameworks for any measurable cost reduction rather than requiring specific savings thresholds. This low barrier for cost justification further validates the market need for real-time FinOps solutions.

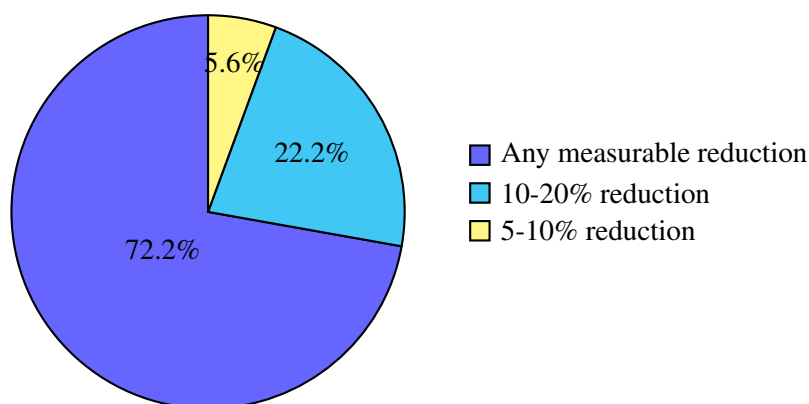


Figure 4.4: Framework cost reduction expectations: 72.2% of respondents would accept any measurable reduction, while 22.2% expect a 10-20% reduction, and 5.6% expect a 5-10% reduction in cloud costs.

The findings from the previous two figures indicate that the proposed frameworks have strong adoption potential, with an elevated interest in their core ideas.

On top of that, the survey results also gave insights into the alignment between the identified challenges and the planned solution capabilities, as well as some considerations that should be taken into account for a successful implementation:

Feature-Problem Alignment: Prevention of unexpected cost spikes received unanimous high ratings, directly matching the most critical pain point identified earlier. This convergence

between problems and solutions strengthens the framework's value proposition and confirms that the planned capabilities address genuine user needs.

Implementation Considerations: Security and compliance concerns represent the primary barrier for sixty-one percent of organizations, providing essential guidance for deployment planning and integration requirements. This finding suggests that careful attention to security architecture will be critical for successful adoption.

This alignment between the identified challenges and the proposed solutions indicates that the frameworks are well-positioned to address real-world problems faced by cloud tenants, providing a solid foundation for their development and implementation.

4.6.4 Implications for Research Development

From the analysis of the survey results, several acknowledgments can be made that will guide the development of the frameworks and their features:

Feature Prioritization: The survey results provide clear guidance for feature prioritization, with real-time cost spike prevention emerging as the most critical capability. The manual analysis burden affecting over 70% of organizations indicates that automation features should be emphasized throughout the development process.

Integration Requirements: The identification of security and compliance concerns as primary barriers inform the architectural decisions for both frameworks. The emphasis on existing tool compatibility suggests that integration with Prometheus, Grafana, and AlertManager represents a technical choice and a strategic requirement for adoption.

Validation Metrics: The low-cost reduction thresholds required for adoption justification indicate that the frameworks' value proposition extends beyond direct savings to encompass operational efficiency and risk mitigation benefits. This finding will inform the experimental phase's validation methodology and success criteria.

This survey establishes a baseline understanding of user needs and expectations, clearly aligning the identified challenges and the proposed solutions. The results will guide the development of the frameworks, ensuring they address real-world problems and provide value to their users.

The following chapters will detail the design, implementation, and validation of these frameworks, putting into practice the insights gained from this survey to create a solution that effectively meets the needs of cloud tenants. In Chapter 8, a more technical and detailed analysis of the survey results will be presented to validate this dissertation's design choices and provide a more comprehensive understanding of the user demand for the proposed frameworks.

Chapter 5

CloudWatch-based Framework

This chapter presents a comprehensive implementation and evaluation of a FinOps framework designed for **AWS** based on CloudWatch. Despite being developed with extensibility in mind, the current implementation focuses on Amazon **EC2**. Having said this, this chapter provides an overview of the framework’s architecture, as well as the specific algorithms and techniques used to monitor, analyze, and optimize **EC2** instances in real-time.

Section 5.1 establishes the context and motivation to develop this solution. Section 5.2 presents the system architecture, including design principles, core components, and operational modes. Finally, Section 5.3 examines the technical implementation, covering the strategy for metrics collection, alert configuration and management, Prometheus data storage, and algorithms for cost and resource analysis and optimization.

5.1 Introduction	42
5.2 Architecture Overview	43
5.3 Implementation Details	49

The codebase for this framework is available on Github [76].

5.1 Introduction

Computing services, such as Amazon **EC2**, are the foundation of modern cloud computing, being one of the most used services in the cloud. They provide scalable and flexible computing resources without investing or maintaining physical hardware, allowing tenants to focus on their applications and services rather than infrastructure management. However, the complexity of managing these resources can lead to cost inefficiencies due to over-provisioning, underutilization, and lack of visibility into resource utilization and, with the possible delay between resource utilization and cost reporting [72], a cloud tenant may not be aware of this inefficiencies until it is too late, leading to unexpected expenses and possible budget overruns.

The following sections present a **FinOps** framework, which is developed on top of Amazon CloudWatch's Application Programming Interfaces (**APIs**). Currently, this framework only has the implementation for Amazon **EC2** instances; however, it is designed to be easily extensible to other **AWS** services, as described in the following sections. The framework provides real-time cost monitoring, predictive cost forecasting, and actionable recommendations to optimize resource utilization and reduce cloud costs. It leverages Amazon CloudWatch for real-time metrics collection, Prometheus for data storage, and Grafana for visualization.

5.2 Architecture Overview

5.2.1 Design Principles

During the design of this framework, several principles were followed to ensure its effectiveness, usability, and extensibility. Among these principles, the following stand out:

Modularity and Separation of Concerns: The system is decomposed into several independent modules, each responsible for a specific aspect of the framework's pipeline, as shown in Section 5.2.2. This separation enables a framework developer to maintain and extend the system more easily, without understanding the entire codebase. It also allows easier testing and debugging, as each module can be tested independently.

Cost-aware development: Since this framework is built on top of Amazon CloudWatch, and is intended to be used with **AWS** resources, which incur costs, the framework supports the option to use LocalStack [56] to simulate Amazon CloudWatch's **API** features locally, allowing for development and testing without incurring costs.

Real-time Processing and Responsiveness: The framework architecture is designed with real-time processing of large volumes of data in mind, prioritizing low-latency data processing and near-real-time responsiveness. This is achieved by using asynchronous processing techniques.

Standardized Tooling: The framework uses industry-standard tools, such as Prometheus and Grafana, to ensure easy integration with existing ecosystems and benefit from these tools' mature features and community support.

Extensibility and Service Agnostic Design: While the framework currently only supports **EC2** instances, the framework is designed to anticipate future extensions to support other **AWS** services, without the need to rewrite or completely understand the entire framework or other modules, making it so that a **FinOps** practitioner can focus solely on the service to be added and its specific requirements. This is achieved by using abstract base classes and factory patterns [35] to facilitate adding new services without breaking the existing functionalities of other services.

These principles, while not exhaustive, provide a solid foundation for the framework’s design and implementation, ensuring that it is robust, maintainable, and adaptable to future requirements.

5.2.2 System Architecture

The framework follows a pipeline architecture, where data flows through several steps, each responsible for a specific task, as shown in Figure 5.1.

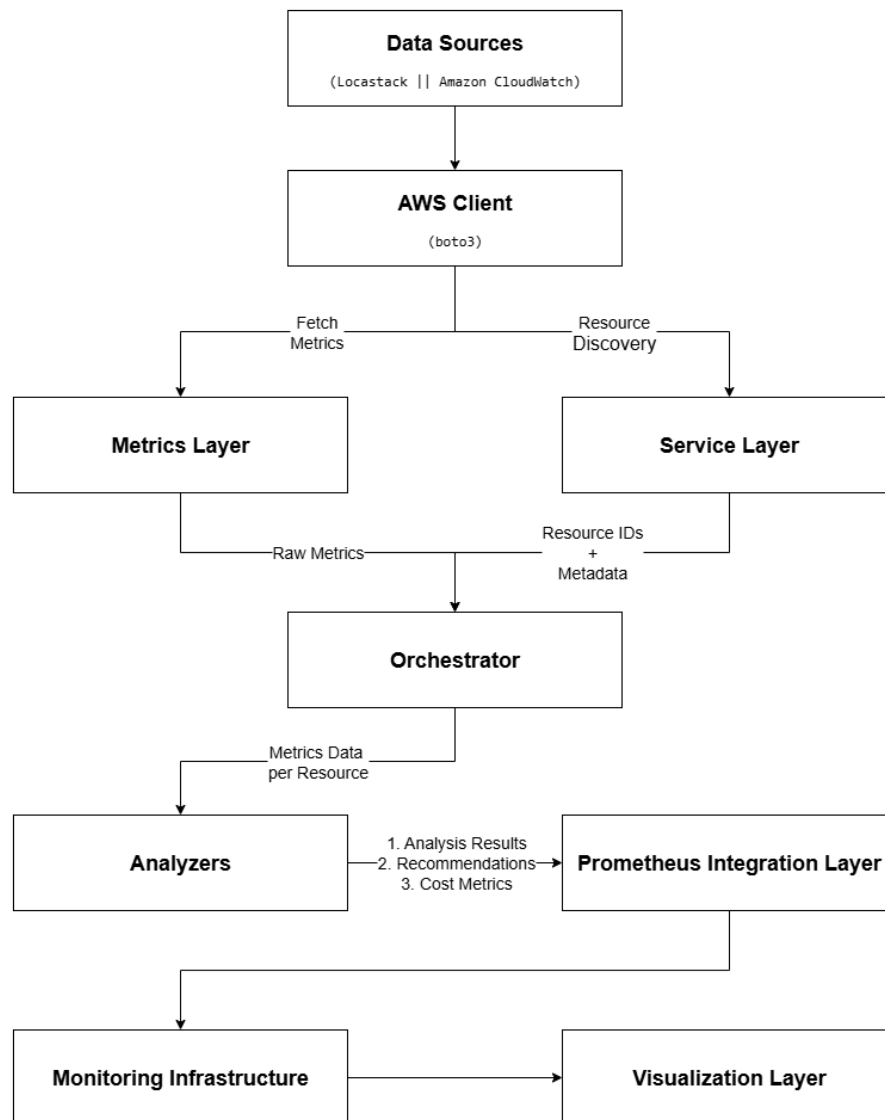


Figure 5.1: CloudWatch-based Framework Architecture Overview

As illustrated in the figure above, the framework can be divided into several loosely coupled modules, each with its own responsibilities:

Data Sources Layer: This layer represents the sources of metrics data for this framework, which can be either Amazon CloudWatch’s **APIs** or LocalStack, depending on the operational

mode of the framework, as described in Section 5.2.3. This layer abstracts the data source, allowing for seamless switching between simulated and real **AWS** environments. **AWS** mode connects to real Amazon CloudWatch **APIs**. Generator and CSV analysis modes use LocalStack to simulate AWS services locally. For CSV analysis mode, metrics are read from exported CloudWatch data files and injected into LocalStack.

AWS Client Layer: Implemented through a singleton pattern [77] to ensure a single instance of the **AWS** client is used throughout the framework, this layer provides an interface for interactions with **AWS** services using boto3 [12]. It manages connections, credential handling, and endpoint configuration based on the operational mode. This layer ensures reliable data retrieval while abstracting the differences between **AWS** services and LocalStack simulations. While this layer does not fetch data directly, it provides authenticated client instances that other layers use to interact with **AWS** services or LocalStack.

Service Layer: The service layer contains **AWS** service-specific implementations that inherit from the abstract base class `AWSService`. This layer is responsible for handling the specifics of each **AWS** service, and provides methods for easy data retrieval and processing. Currently, only the **EC2** service is implemented. It discovers **EC2** instances, retrieves their metadata — such as instance type, states, and tags — and defines service-specific configurations, such as the required metrics to be collected and default mock patterns. Having said this, the framework is designed so that new services can be added easily by inheriting from the abstract base class and implementing the required methods. This layer uses the AWS Client Layer to obtain **EC2** instances and their metadata, but does not directly fetch metrics data.

Metrics Layer: This layer is centered around the `CloudWatchMetrics` class, which is responsible for fetching metrics from Amazon CloudWatch. It uses a CloudWatch client — instantiated by the AWS Client Layer — to fetch metrics such as CPU utilization, memory utilization, and network traffic. Despite not being currently used, this layer supports batch fetching capabilities, allowing for an easier retrieval of multiple metrics simultaneously.

Orchestrator: The Orchestrator is the central coordination component implemented in the main monitoring loop. It orchestrates the data collection and analysis pipeline by: (1) using the Service Layer to discover resources — for this specific implementation, **EC2** instances — and their metadata; (2) invoking the Metrics Layer to collect performance data for each discovered resource; (3) aggregating collected metrics and metadata into a structured format; (4) passing that structured data into the Analysis Engine for evaluation; and (5) forwarding both the raw metrics and the analysis results to the Prometheus Integration Layer. The Orchestrator uses asynchronous processing with Python's AsyncIO library [10] to handle multiple resources concurrently.

Analysis Engine: This module contains the core logic for the cost analysis and optimization recommendation. The `AnalyzerFactory` class uses a factory pattern [35] to instantiate

service-specific analyzers — in this case, the `EC2Analyzer` class. The analyzer receives aggregated metrics and metadata from the Orchestrator and processes them to calculate efficiency scores based on resource utilization, identify underutilized or overprovisioned instances, generate cost optimization recommendations — such as downsizing or terminating instances — and compute potential monthly cost savings. This analysis considers multiple factors, including CPU and memory utilization, network traffic, and instance pricing data. More details on the analysis algorithms can be found in Section 5.3.

Prometheus Integration Layer: This layer transforms the analysis results and raw metrics into a format suitable for Prometheus. This layer manages the connection to the Prometheus server and handles metrics registration and the sanitization of metric names, labels, and values. It also provides methods for pushing metrics to Prometheus. This allows developers to work with the framework without worrying about Prometheus internals, as it hides the complexity of interacting with it behind a few idiomatic and straightforward methods.

Monitoring Infrastructure: This layer refers to the actual Prometheus server and AlertManager, which are used to store and retrieve metrics, and to manage alerts displayed to the user, respectively. Prometheus gathers metrics through the Prometheus Integration Layer, and AlertManager processes previously defined alert rules to notify users of the framework of cost optimization opportunities or resource inefficiencies.

Visualization Layer: This layer, which is built on top of Grafana, provides interactive dashboards for visualizing metrics, costs, and recommendations. It includes pre-configured dashboards, which can be seen in Appendix C.2, for cost overview, resource utilization, efficiency scores, and alerts. This layer is designed to be user-friendly, allowing end users of the framework to easily monitor their cloud resources and costs without needing to understand the underlying complexities of the framework.

This modular architecture promotes a clear separation of concerns, making it easier to maintain, extend, and test the framework. Each module can be developed and tested almost independently, allowing for a more agile development process.

5.2.3 Operational Modes

This framework supports three distinct operation modes to support various use cases: (1) Generator mode, which simulates metrics for `EC2` instances according to configurable profiles, which are passed to the framework by the user. This mode is particularly useful for development and testing purposes, as it allows users to simulate different workload patterns without needing actual `AWS` resources; (2) CSV analysis mode, which is essentially a discrete simulation mode that processes exported `AWS` logs and populates Prometheus and LocalStack [56] on a regular basis. This mode is mostly useful for users with a running `AWS` infrastructure and who want to analyze historical data without running the framework or spending money on `AWS`. It is also useful for testing the framework's algorithms against real-world data; and (3) `AWS` mode, where the framework

connects directly to Amazon CloudWatch’s **APIs** to collect real-time metrics from actual **AWS** infrastructure. This is intended for production deployments where monitoring of live resources is essential.

Both Generator and CSV Analysis modes use LocalStack [56] to simulate Amazon CloudWatch’s **APIs** functionalities locally, allowing for development, testing, and demonstration without actually paying for **AWS** services.

5.2.4 Configuration Management

The framework has a configuration system to manage various settings. This system separates global settings, service-specific settings, and resource specifications. These configurations are stored across multiple YAML files (refer to Appendix B.2.2) to promote modularity and ease of maintenance. The base configuration file defines global parameters such as monitoring frequency and Prometheus port. Service-specific configurations are implemented as separate YAML files, allowing different services to have their settings in accordance with their specific requirements. Service-specific configurations define metrics namespaces, required metrics for each service type, mock profile definitions, and analysis thresholds. Depending on the operational mode and the flags passed to the framework, the framework will use the appropriate configuration values, such as the mock profiles for the Generator mode.

Instance specifications are stored in a separate JSON file containing information for each instance type, such as vCPUs, memory, and cost per hour. The framework uses this file to calculate pricing and generate recommendations based on the instance type and specifications.

For example, the specification for an `t2.micro` instance on `eu-central-1` is as follows:

Listing 5.1: EC2 Instance Specification Example

```
1 {  
2   "t2.micro": {  
3     "vcpus": 1,  
4     "memory": 1,  
5     "cost_per_hour": 0.0116  
6   }  
7 }
```

This separation allows for new instance types to be added easily, and for a user to define their instance types and pricing — which may vary depending on the region and the Service Level Agreement (**SLA**) — without needing to modify the framework’s codebase. Furthermore, this format allows for future extensions where this information can be automatically scraped from **AWS**. New information can be added to the instance specifications and easily integrated into the framework, such as storage options, reserved and spot instance pricing, and other relevant information.

5.2.5 Resource Definition and Limits

AWS considers that the most critical metrics for rightsizing recommendations are CPU utilization, memory utilization, network traffic, disk I/O, and performance of attached EBS volumes [89]. For the sake of simplicity and due to the limited time, our framework focuses on the first three metrics — CPU, memory, and network traffic.

These resources are then used on the framework’s analysis algorithms to determine efficiency scores and generate recommendations. According to AWS documentation, an instance is idle if peak CPU utilization is below or equal to 1% over 14 days. At the same time, underutilization occurs when CPU utilization exceeds 1% but still leaves optimization potential [89]. Azure advisors classify an instance as underutilized if the average CPU utilization over four or more days is below or equal to 5% [44].

During the development and testing of our framework, we found that these thresholds were too conservative for our use case, and, since the aim of the framework is to provide actionable recommendations, we decided that a more aggressive approach would be more suitable for our framework. This is justifiable because even if false positives occur, the framework will still provide recommendations that someone with technical knowledge of the system can manually review. Furthermore, due to the configurability of the framework, users can easily adjust these thresholds to suit their specific needs and operational requirements. With this in mind, this framework adopts a 5% CPU threshold for idle detection and 20% for underutilization.

Memory is bundled into the instance costs and, by default, does not incur additional charges in EC2 instances. However, monitoring memory utilization is still very important for identifying overprovisioned instances, calculating efficiency scores, and recommending rightsizing actions, particularly for memory-intensive workloads like databases. During the development of the framework, we found that for our use cases, memory utilization tended to be more stable than CPU utilization, and therefore decided to use a 10% threshold for idle detection and 30% for underutilization.

Network traffic is a more complex metric to analyze, as it can vary significantly depending on the workload and the instance type. AWS Trusted Advisor identifies instances as underutilized when network I/O falls below 5 MB for extended periods [29]. However, network utilization patterns can widely vary based on workload characteristics — e.g., web servers may experience high burst traffic, while databases may have a more consistent traffic pattern. For this framework, a 1% network bandwidth utilization threshold was set for idle detection and a 10% one for underutilization. These thresholds are calculated relative to the maximum network bandwidth available for each instance type, ensuring fair comparison across different instance families that offer different network performance capabilities.

5.2.6 Resource Weight Distribution

This framework employs a weighted scoring system to calculate efficiency scores for each instance based on its resource utilization, where the weights are configurable by the user in the EC2 con-

figuration file. The weights that were used during this research are as follows: (1) CPU utilization is set to 40%, (2) Memory utilization is set to 30%, and (3) Network traffic is set to 30%.

The 40% weight assigned to CPU utilization reflects that CPU is often the baseline metric most administrators use for measuring resource utilization, with more experienced managers also accounting for memory consumption [30].

Memory is assigned a 30% weight, recognizing its importance in performance and cost optimization, especially for memory-intensive workloads such as databases and in-memory caches.

Network traffic is also assigned a 30% weight, as it can significantly impact costs and performance, because despite not being directly charged in the **EC2** pricing model, it can influence the overall cloud costs, especially when considering data transfer between instances or across **AZs**.

Having said this, for memory metrics to be collected, the CloudWatch agent must be installed on all the desired instances, unlike CPU metrics, which are collected by default [8]. Having said this, for cases where memory metrics are unavailable, the framework will still calculate efficiency scores based on CPU and network traffic, giving a weight of 55% to CPU and 45% to network traffic, keeping the proportions of the original weights. This ensures the framework can still provide valuable insights even when memory metrics are unavailable.

5.3 Implementation Details

This section focuses on the technical implementation of the framework, focusing on the core algorithms and architectural decisions of the framework.

5.3.1 Metric Collection Strategy

The framework implements an asynchronous metric collection pipeline that concurrently gathers data from multiple **EC2** instances. This collection process is done on a configurable interval — set to 60 seconds by default — and features a retry mechanism to handle possible failures during the metric collection process.

As previously mentioned, this is powered by Python's AsyncIO library, which allows the framework to monitor various instances simultaneously without blocking individual metric retrievals. This approach is essential for maintaining low latency in data collection, especially for our use case, since we are dealing with real-time metrics that can change rapidly.

Firstly, the framework discovers all active **EC2** instances using the AWS Service Layer. For each discovered instance, it creates a task that retrieves CloudWatch metrics for CPU utilization, memory utilization, and network traffic. If any failure occurs, the framework retries the metric collection up to three times before logging an error and moving on to the next instance.

Once metrics are collected, they are placed into a dictionary, where the keys are the instance IDs and the values are dictionaries containing the metrics data for each instance. Among these are raw metrics from CloudWatch and computed statistics such as mean, minimum, and maximum values over the collection period. This data structure is then passed to the Analysis Engine for

further processing, where it is used to calculate metrics, efficiency scores, and generate recommendations.

5.3.2 Alert Configuration and Monitoring

The framework uses Prometheus AlertManager [4] to provide real-time alerts and notifications based on the collected metrics and analysis results.

Each alert is configured with a specific evaluation period and thresholds to optimize the balance between rapid detection and false positive reduction. Critical alerts, such as cost spikes, have shorter evaluation periods to ensure that they are detected and acted upon as fast as possible. Meanwhile, trend-based alerts, such as week-over-week cost increases, use extended periods to identify patterns over time.

Some of these alerts include rich annotations with variables to provide context and actionable information, as can be seen in Figure C.14 — a Grafana dashboard showing the alerts that are currently active in the framework.

Alerts are defined in a YAML configuration file, which specifies the alert rules, conditions, time to trigger, labels, annotations, and severity levels. The complete alert configuration, including all rule definitions and explanations of each alert's purpose and behaviour, is documented in Appendix E.2.

5.3.3 Data Storage in Prometheus

This framework uses Prometheus to store and retrieve metrics data. The Prometheus Integration Layer manages the transformation of collected metrics into a format suitable for Prometheus, including sanitization of metric and label names, handling metrics registration, and pushing metrics to Prometheus.

This layer consists of a Prometheus Wrapper class, which provides a simple, yet effective interface for users of the framework to interact with Prometheus without worrying about the conventions defined by Prometheus.

Moreover, this layer implements a thread-safe singleton pattern [78] for managing Prometheus gauges and info metrics, ensuring these are consistent across the framework's lifecycle. Each metric can be registered with a unique name and labels, allowing users to query and aggregate metrics easily.

5.3.4 Algorithms Overview

This section describes the implementation of the algorithms used for cost and resource monitoring, analysis, and optimization in the framework. Most of these algorithms relate to the Analysis Engine mentioned in Section 5.2.2.

5.3.4.1 Efficiency Score Calculation

The efficiency score is a key metric used to evaluate the performance and cost-effectiveness of EC2 instances. It is calculated based on the collected metrics, using a weighted scoring system that considers CPU utilization, memory utilization, and network traffic.

Let CPU_{util} , Mem_{util} , Net_{in} , and Net_{out} represent the collected CPU, memory, and network metrics over the evaluation period, and let $W = \{w_{cpu}, w_{mem}, w_{net}\}$ denote the configured weights for those same metrics, respectively. Since the weights are configurable by the user, the following condition must hold to ensure that the algorithm runs correctly:

$$\sum_{k \in \{cpu, mem, net\}} w_k = 1 \quad (5.1)$$

Metric Normalization The algorithm starts by calculating the mean values for CPU and memory utilization, and then divides these values by 100 to normalize them to the range $[0, 1]$:

$$\text{CPU}_{norm} = \frac{\text{mean}(\text{CPU}_{util})}{100} \quad (5.2)$$

$$\text{Mem}_{norm} = \frac{\text{mean}(\text{Mem}_{util})}{100} \quad (5.3)$$

Network utilization is not as direct as the previous two metrics, as it is not a percentage but rather a raw value. Having said that, the algorithm computes the total average throughput and normalizes it against the maximum observed throughput for the specific instance type:

$$\text{Net}_{total} = \text{mean}(\text{Net}_{in} + \text{Net}_{out}) \quad (5.4)$$

$$\text{Net}_{norm} = \begin{cases} \min\left(\frac{\text{Net}_{total}}{\text{Net}_{max}(i_j)}, 1\right) & \text{if } \text{Net}_{max}(i_j) > 0 \\ 0 & \text{otherwise} \end{cases} \quad (5.5)$$

where $\text{Net}_{max}(i_j)$ represents the maximum observed network throughput for instance type $i_j \in I$, knowing that $I = \{i_1, i_2, \dots, i_n\}$ is the set of all instance types. The $\min$ operator ensures the normalized value does not exceed 1 in cases where current usage is higher than the previously observed maximum.

Weighted Score Computation The efficiency score E is then calculated as a weighted linear combination of the normalized utilization metrics:

$$E' = 100 \times (w_{cpu} \cdot \text{CPU}_{norm} + w_{mem} \cdot \text{Mem}_{norm} + w_{net} \cdot \text{Net}_{norm}) \quad (5.6)$$

To ensure the score remains within valid bounds, the final efficiency score is kept within the range $[0, 100]$:

$$E = \max(0, \min(100, E')) \quad (5.7)$$

This scoring mechanism provides a standardized metric for comparing instance efficiency across different configurations and workloads, with higher scores indicating better resource utilization and lower scores meaning that there may be over-provisioning and opportunities for resource optimization. The configurable weights allow users of the framework to prioritize different resource dimensions according to their specific workload characteristics and goals.

Algorithm 7 represents this algorithm in pseudocode form.

5.3.4.2 Recommendation Generation Algorithm

The recommendation generation algorithm analyzes the efficiency score and resource metrics to generate actionable recommendations for cost optimization. Algorithm 1 details this process.

The recommendation generation algorithm analyzes the efficiency score and resource metrics to generate actionable recommendations for cost optimization. The algorithm follows a structured decision-making process that evaluates instance utilization — through the efficiency score — against user configurable thresholds to determine appropriate optimization strategies.

Algorithm 1 details this process through an approach that separates the main decision logic from the specific recommendation construction procedures. This decomposition is used for readability purposes.

Algorithm 1 Generating Termination and Rightsizing Recommendations Based on Efficiency Scores and Usage Patterns

```

1: procedure GENERATERECOMMENDATIONS(instance, metrics, efficiency_score)
2:   recommendations  $\leftarrow$  []
3:   monthly_cost  $\leftarrow$  CALCULATEMONTHLYCOST(instance.hourly_cost)
4:    $\triangleright$  Identify inefficiency causes for reporting
5:   causes  $\leftarrow$  IDENTIFYINEFFICIENCYCAUSES(metrics)
6:   if efficiency_score < thresholds.very_low then
7:     rec  $\leftarrow$  CREATETERMINATIONREC(monthly_cost)
8:     recommendations.append(rec)
9:   else if efficiency_score < thresholds.low then
10:    rightsizing_rec  $\leftarrow$  CREATERIGHTSIZINGREC(instance, metrics, monthly_cost, causes)
11:    if rightsizing_rec  $\neq$  null then
12:      recommendations.append(rightsizing_rec)
13:    end if
14:  end if
15:  return recommendations
16: end procedure

```

As shown in Algorithm 1, for instances with extremely low efficiency scores, the algorithm generates termination recommendations, as shown in Algorithm 2.

Algorithm 2 Create Termination Recommendation

```

1: procedure CREATETERMINATIONREC(monthly_cost)
2:   rec  $\leftarrow$  {
3:     type: "terminate",
4:     monthly_savings: monthly_cost,
5:     reason: "Instance appears to be idle"
6:   }
7:   return rec
8: end procedure

```

For instances with low efficiency scores the algorithm generates rightsizing recommendations, as shown in Algorithm 3. In this algorithm, the framework starts by determining the optimal instance type, as detailed in Section 5.3.4.3. If an optimal instance type is found, the algorithm calculates the potential monthly savings by comparing the current instance's monthly cost with the optimal instance's monthly cost. If the savings exceed a user-defined threshold, a rightsizing recommendation is created and returned. Otherwise, no recommendation is generated.

Algorithm 3 Create Rightsizing Recommendation

```

1: procedure CREATERIGHTSIZINGREC(instance, metrics, monthly_cost, causes)
2:   optimal  $\leftarrow$  FINDOPTIMALINSTANCE(instance, metrics)
3:   if optimal = null then
4:     return null
5:   end if
6:   new_cost  $\leftarrow$  GETMONTHLYCOST(optimal)
7:   savings  $\leftarrow$  monthly_cost - new_cost
8:   if savings  $\geq$  thresholds.min_savings then
9:     rec  $\leftarrow$  BUILDRIGHTSIZINGOBJECT(optimal, savings, causes)
10:    rec  $\leftarrow$  {
11:      type: "downsize",
12:      suggested_type: optimal,
13:      monthly_savings: savings,
14:      reason: causes
15:    }
16:    return rec
17:   else
18:     return null
19:   end if
20: end procedure

```

Another component of the recommendation generation algorithm is the identification of specific inefficiencies, which provides more context to the recommendations. Since the efficiency score is a weighted sum of the normalized metrics, even if the efficiency score is low, it does not mean that all metrics are low. Therefore, the auxiliary function detailed in Algorithm 4 is used to analyze these metrics and provide more context to the recommendations.

Algorithm 4 Threshold-Based Detection of CPU, Memory, and Network underutilization with severity categorization

```

1: procedure IDENTIFYINEFFICIENCYCAUSES(normalized_metrics, raw_metrics)
2:   causes  $\leftarrow$  []
3:                                      $\triangleright$  Extract threshold values from configuration
4:   cpu_thresholds  $\leftarrow$  config.utilization_ratio.cpu
5:   memory_thresholds  $\leftarrow$  config.utilization_ratio.memory
6:   network_thresholds  $\leftarrow$  config.utilization_ratio.network
7:                                      $\triangleright$  Check CPU utilization levels
8:   if normalized_metrics.cpu < cpu_thresholds.very_low then
9:     causes.append(f"extremely low CPU utilization: {raw_metrics.cpu}%")
10:  else if normalized_metrics.cpu < cpu_thresholds.low then
11:    causes.append(f"low CPU utilization: {raw_metrics.cpu}%")
12:  end if
13:                                      $\triangleright$  Check memory utilization levels
14:  if normalized_metrics.memory < memory_thresholds.very_low then
15:    causes.append(f"extremely low memory utilization: {raw_metrics.memory}%")
16:  else if normalized_metrics.memory < memory_thresholds.low then
17:    causes.append(f"low memory utilization: {raw_metrics.memory}%")
18:  end if
19:                                      $\triangleright$  Check network activity levels
20:  network_mbps  $\leftarrow$   $\frac{\text{raw\_metrics.total\_network}}{1024^2}$ 
21:  if normalized_metrics.network < network_thresholds.very_low then
22:    causes.append(f"minimal network activity: {network_mbps} MB/s")
23:  else if normalized_metrics.network < network_thresholds.low then
24:    causes.append(f"low network activity: {network_mbps} MB/s")
25:  end if
26:  return causes
27: end procedure

```

This algorithm checks each normalized resource metric against the configured thresholds to generate a list of causes for the inefficiency score. The algorithm distinguishes between low and very low utilization levels to provide more granular insights into the resource utilization patterns. Network activity is converted from bytes to megabytes per second for more straightforward interpretation, while CPU and memory utilization are reported as percentages.

5.3.4.3 Optimal Instance Selection

The algorithm for determining optimal EC2 instance configurations can be described as a multi-stage optimization process. Algorithm 8 presents the pseudocode for this algorithm.

Before explaining the algorithm, it is important to define the variables used in the algorithm:

Let $I = \{i_1, i_2, \dots, i_n\}$ be the set of available instance types, where each instance $i_j \in I$ contains vCPU_{i_j} virtual CPUs, Mem_{i_j} Gigabytes (GBs) of memory, and costs Price_{i_j} dollars per hour. This structure represents the instance catalog, which is defined in the configuration file, as described in Section 5.2.4.

Let i_{current} be the current instance under evaluation with specifications $\text{vCPU}_{\text{current}}$, $\text{Mem}_{\text{current}}$, and $\text{Price}_{\text{current}}$.

Let CPU_{util} be the observed CPU utilization during the evaluation period, Mem_{util} the observed memory utilization, Net_{in} be the observed incoming network traffic, and Net_{out} the observed outgoing network traffic.

Let $\beta = 1.3$ be the safety margin factor, which is a user-configurable buffer to account for unexpected spikes in resource utilization.

Resource Requirement Calculation The first step in the algorithm is to calculate the required resources based on observed utilization patterns:

$$\text{vCPU}_{\text{required}} = \left\lceil \frac{\text{mean}(\text{CPU}_{\text{util}})}{100} \times \text{vCPU}_{\text{current}} \times \beta \right\rceil \quad (5.8)$$

$$\text{Mem}_{\text{required}} = \left\lceil \frac{\text{mean}(\text{Mem}_{\text{util}})}{100} \times \text{Mem}_{\text{current}} \times \beta \right\rceil \quad (5.9)$$

$$\text{Net}_{\text{required}} = \text{mean}(\text{Net}_{\text{in}} + \text{Net}_{\text{out}}) \times \beta \quad (5.10)$$

The ceiling function ensures that the fraction's resources are rounded up to the nearest whole number, as partial resources are not applicable in the context of EC2 instances.

Instance Feasibility Check For each instance $i_j \in I$, the algorithm evaluates its feasibility based on four criteria. The set of feasible instances $\mathcal{F}$ is defined as:

$$\forall i_j \in I : i_j \in \mathcal{F} \iff \left(\begin{array}{l} i_j \neq i_{\text{current}} \wedge \\ \text{vCPU}_{i_j} \geq \text{vCPU}_{\text{required}} \wedge \\ \text{Mem}_{i_j} \geq \text{Mem}_{\text{required}} \wedge \\ (\nexists \text{Net}_{\text{max}}(i_j) \vee \text{Net}_{\text{max}}(i_j) \geq \text{Net}_{\text{required}}) \end{array} \right) \quad (5.11)$$

Where $\text{Net}_{\text{max}}(i_j)$ represents the maximum observed network throughput for instance type i_j based on historical data. The last condition ensures that either no historical network data exists for the instance type — allowing new instance types to be considered — or, if it does exist, that the instance type has enough network capacity to handle the expected network traffic.

Cost-Optimal Selection Once the set of feasible instances $\mathcal{F}$ is determined, the algorithm proceeds to identify the instance $i_{optimal} \in \mathcal{F}$ with the lowest hourly cost:

$$i_{optimal} = \underset{i_j \in \mathcal{F}}{\operatorname{argmin}} \operatorname{Price}_{i_j} \quad (5.12)$$

If the feasible set is empty, the algorithm returns null, indicating that no instance meets the requirements.

Recommendation Generation The last step of the algorithm involves checking whether to generate a recommendation based on the potential cost savings. The algorithm produces a recommendation when:

$$\text{Recommended Instance} = \begin{cases} i_{optimal} & \text{if } \mathcal{F} \neq \emptyset \wedge \operatorname{Price}_{i_{optimal}} < \operatorname{Price}_{current} \\ \emptyset & \text{otherwise} \end{cases} \quad (5.13)$$

This approach ensures that recommendations are generated only when they provide cost benefits while maintaining or improving performance. By requiring that $\operatorname{Price}_{i_{optimal}} < \operatorname{Price}_{current}$, the algorithm avoids recommending instances that would incur possible migration costs without providing any cost savings.

This algorithm is designed to provide workload stability through the safety margin β while systematically identifying opportunities for cost optimization. Algorithm 8 represents this algorithm in pseudocode form.

5.3.4.4 Cost Calculation Methodology

This framework employs a cost calculation methodology that considers the varying number of days each month, rather than using simplified approximations — such as assuming 30 days per month — to provide more accurate cost estimates. The hourly costs are retrieved from the instance specifications JSON file described in Listing 5.1.

The monthly cost for an EC2 instance is calculated using the following formula:

$$C_{monthly} = C_{hourly} \times 24 \times D_{month} \quad (5.14)$$

Where $C_{monthly}$ is the total monthly cost, C_{hourly} is the hourly cost of the given instance type, and D_{month} is the number of days in the specific month.

The framework uses Python's `calendar.monthrange()` method to determine the exact number of days for any given month and year, ensuring accurate cost calculations. For example, February costs will be calculated using 28 or 29 days, while January will use 31, providing more precise savings estimates for the user than the standard approximation of 730 hours per month. While for our experiments, this approach did not yield significant differences in the results, for

users with long-term monitoring and volatile workloads, this can lead to considerable cost differences over time.

5.3.4.5 Saving Calculation Methodology

The cost calculation methodology presented in Section [refsubsubsec:ec2-predictive-cost-forecasting](#) extends to calculating potential savings when recommending action such as downsizing or terminating instances. These savings are calculated like this:

$$S_{monthly} = C_{current}^{monthly} - C_{recommended}^{monthly} \quad (5.15)$$

Where $S_{monthly}$ is the potential monthly savings, $C_{current}^{monthly}$ is the current instance's monthly cost, and $C_{recommended}^{monthly}$ is the recommended instance's monthly cost.

For termination recommendations, $C_{monthly}^{recommended} = 0$, making the savings equal to the full monthly cost of the instance.

Chapter 6

Kubernetes-based Framework

This chapter presents the comprehensive implementation and evaluation of a FinOps framework for Kubernetes.

Section 6.1 establishes the theoretical foundation and motivation behind this framework. Section 6.2 presents the framework’s system architecture and design principles. Finally, Section 6.3 analyzes the technical implementation details, such as the algorithms used for cost and resource analysis and optimization.

6.1	Introduction	58
6.2	Architecture Overview	59
6.3	Implementation Details	62

The codebase for this framework is available on Github [42].

6.1 Introduction

Kubernetes has become increasingly popular for managing and orchestrating containerized applications, with its ability to automate deployment, scaling, and operations in distributed environments. With this increasing popularity, alternatives for the creation and management of Kubernetes clusters on the cloud have also emerged, such as Amazon **EKS**, Google **GKE**, and Azure **AKS**. However, with the inherent complexity of Kubernetes, having control and granular visibility over its associated costs can be challenging. This is where this framework comes into play, providing a comprehensive solution for FinOps in Kubernetes environments and allowing organizations to effectively observe, analyze, and optimize their cloud costs in real-time. The following sections will detail this framework’s architecture, implementation, and algorithms.

6.2 Architecture Overview

6.2.1 Namespace-Level Overview

Figure 6.1 illustrates the architecture of the Kubernetes-based Framework on a namespace level basis, showcasing the various components and their interactions. For the sake of simplicity, the pooling and data requests are not represented. Instead, only the data flow appears. For example, we represent data flowing to Prometheus without a request, as if components published data directly to Prometheus. In reality, Prometheus uses a pull-based model where a ServiceMonitor is configured so that Prometheus can scrape data from the specified targets.

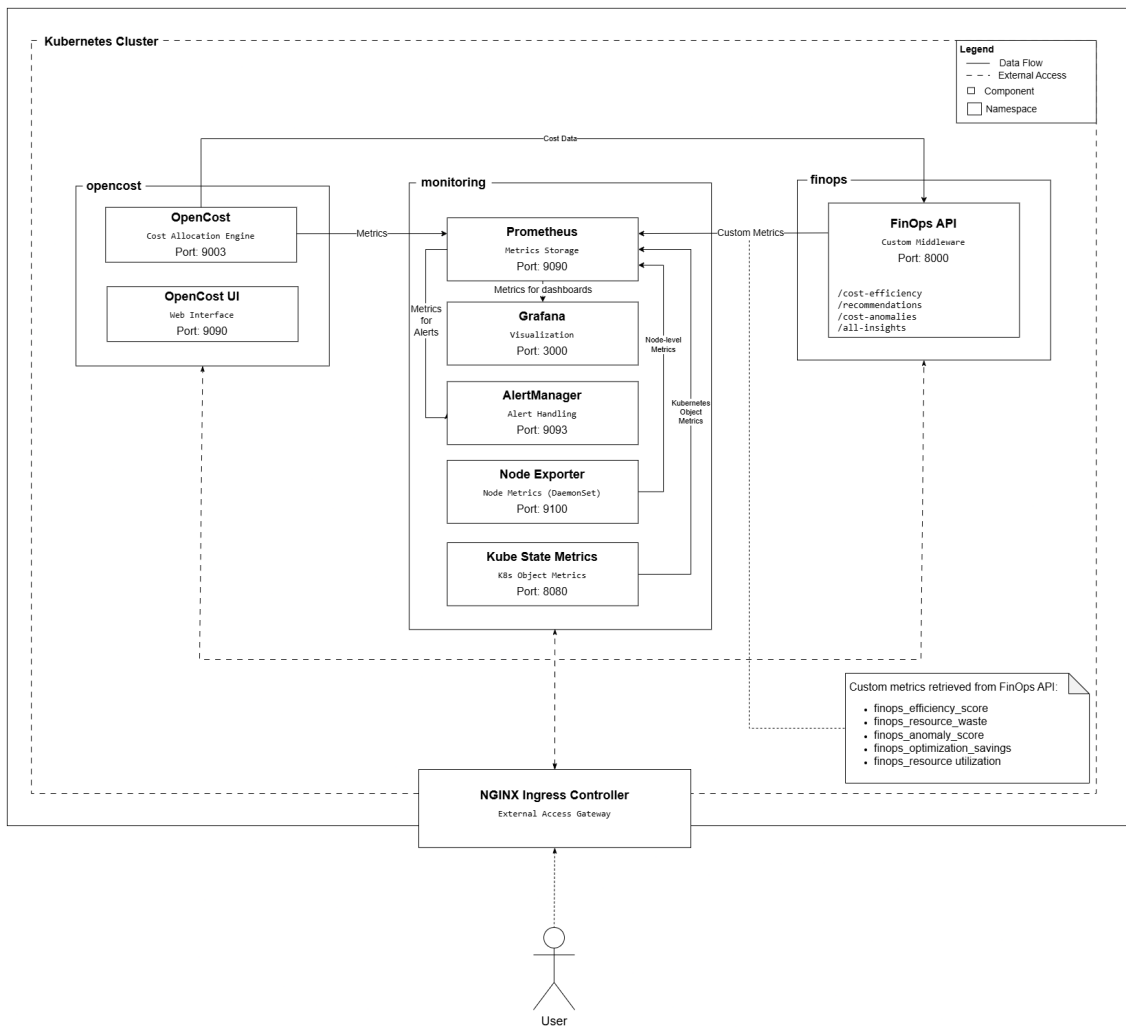


Figure 6.1: Kubernetes-based Framework Architecture Overview - Namespace Level

The diagram above provides a high-level overview of the framework's namespaces, components, and their interaction. The following components are present in the architecture:

Opencost: OpenCost provides real-time insights into cloud infrastructure costs, designed for Kubernetes environments. It serves as the primary data source for cost-related information in

this framework, connecting to cloud billing **APIs** to retrieve pricing information and to Kubernetes and Prometheus to collect resource utilization metrics in real-time. By providing this data, OpenCost enables this framework to calculate real-time costs and to predict future costs based on current resource consumption and known pricing rates, as presented in Section 5.3.4.4.

Opencost UI: The OpenCost UI component provides a web-based interface for users to visualize and interact with the cost data collected by OpenCost. It allows users to explore real-time cost metrics, analyze patterns, and gain more insight into their Kubernetes costs without additional tools or configuration.

Prometheus: Prometheus serves as the primary data store, implementing a pull-based metrics collection model to gather data from various components in the Kubernetes cluster. ServiceMonitors are used to configure Prometheus to scrape metrics from specified targets, ensuring that all relevant data is collected and stored for later usage.

Grafana: Grafana is used for data visualization. It serves as the primary interface for users of the framework to interact with the data, having a pre-configured set of dashboards, as shown in Appendix C.1, which allows users to visualize and interact with the framework's data and gain insights into their Kubernetes costs and resource utilization.

Alertmanager: AlertManager handles notifications and alerts. The framework includes a set of pre-configured alerts that notify users about cost anomalies, resource waste, cost-efficiency issues, and other scenarios that may require attention. The alerts are generated based on data collected from Prometheus, which may be collected from OpenCost, Node Exporter, FinOps API, and Kube State Metrics. These alerts, as well as a brief description of each of them, are presented in Appendix E.1.

FinOps API: The FinOps API is a custom **API**, implemented using Python's FastAPI framework [36], which contains the framework's core logic and algorithms. This **API** handles any logic that cannot be done directly using Prometheus Query Language (**PromQL**) queries and requires more complex calculations, such as the efficiency score calculation, resource waste calculation, and rightsizing recommendations. Furthermore, by having this middleware, the framework can be extended to support additional features and functionalities in the future, and queries can be made simpler and more readable, as **PromQL** queries can be abstracted away from the user, allowing them to focus on the business logic rather than the underlying data retrieval mechanisms. This framework performs this analysis by querying both Prometheus and OpenCost's **API** with the necessary parameters, and then uses that data on various algorithms, as described in Section 6.3.

Node Exporter: Node Exporter is a monitoring agent that runs on each cluster node via a DaemonSet, ensuring that every node is monitored. A DaemonSet is a Kubernetes resource that automatically deploys a copy of a pod on each node of the cluster, ensuring that Node

Exporter runs and collects metrics from all nodes in the cluster. Node Exporter collects hardware and operating system metrics from the nodes in the cluster, such as CPU utilization, memory utilization, disk usage, network traffic, temperature, number of processes, and other system-level metrics that are essential for monitoring the health and performance of the nodes in the cluster. These metrics are then exposed to Prometheus for scraping, allowing the framework to monitor the health and performance of the nodes in the cluster.

Kube State Metrics: Kube State Metrics is a Kubernetes component that exposes the state of Kubernetes objects, such as pods, deployments, services, and nodes, as Prometheus metrics. Unlike Node Exporter, it does not provide metrics about the underlying hardware or operating system, but instead exposes the desired and actual state of Kubernetes resources, such as: (1) which pods are ready and which are not; (2) which deployments are missing replicas; (3) which **PVCs** are bound; (4) what resources requests and limits are set for each container; and (5) the status of nodes, namespaces, and other Kubernetes objects.

NGINX Ingress Controller: The Ingress controller is a component that manages external access to services inside the Kubernetes cluster. It provides load balancing, SSL termination, and routing capabilities based on hostnames and paths to the services running in the cluster. NGINX is one of the most popular Ingress controllers, and was chosen for this framework due to its community support and previous experience with it.

These components work together to provide a comprehensive **FinOps** solution for Kubernetes environments, allowing users to monitor, analyze, and optimize their cloud costs in real-time.

6.2.2 Node-Level Overview

Figure 6.2 provides an overview of the framework's architecture at the node level, showing how the components interact with each other and the Kubernetes cluster. Similarly to the previous figure, the arrows represent the data flow between the components, and not the pooling and data requests made by Prometheus to scrape the metrics from the targets periodically.

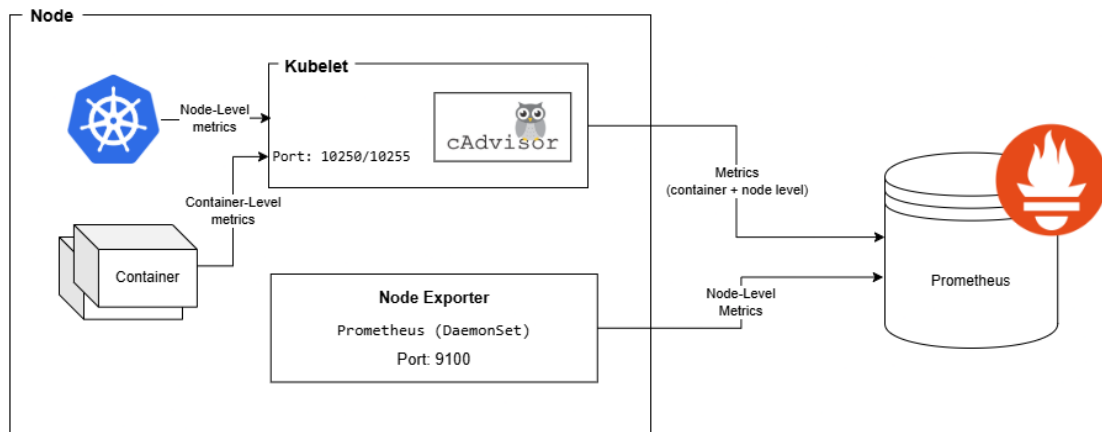


Figure 6.2: Kubernetes-based Framework Architecture Overview - Node Level

The diagram above provides a high-level overview of the framework’s components and their interactions at the node level. Node Exporter — prometheus exporter for hardware and operating system metrics — runs as a DaemonSet on each node of the cluster, ensuring that all nodes are monitored and their metrics are collected by Prometheus, as described in Section 6.2.1.

Kubelet is the main agent that runs on every node in a Kubernetes cluster, responsible for: (1) managing — starting, stopping, and monitoring — containers on the local node; (2) communicating with the control plane to find out which pods should be running on the node; (3) monitoring the health and status of pods, and reporting this information back to the control plane; and (4) exposing metrics — including cAdvisor and Kubelet statistics — on endpoints that Prometheus can scrape. Essentially, without Kubelet, Kubernetes is unable to manage or monitor workloads on the nodes.

cAdvisor — short for Container Advisor — is a component embedded in Kubelet that collects and exports resource utilization metrics for containers running on the given node. It provides metrics such as CPU utilization per container, memory utilization, disk I/O, network traffic, and container lifetime statistics. Tools like Prometheus then use these metrics to monitor the performance and resource utilization of containers in the cluster. Since cAdvisor is integrated into Kubelet, no separate installation is required.

6.3 Implementation Details

This Section focuses on the technical implementation of the framework and the core algorithms used for cost and resource monitoring, analysis, and optimization in the framework.

6.3.1 Data Storage and Retention Strategy

This framework’s data storage and retention strategy is designed to ensure that the data persists for a sufficient period to allow for meaningful analysis and insights.

For this framework, **PVs** were configured for components that require data persistence, specifically Grafana and Prometheus. The use of **PVs** triggers Kubernetes to create StatefulSets instead of standard Deployments, providing stable network identities and guaranteeing ordered deployment sequences.

Prometheus is configured to use a **PV** for data storage, ensuring the collected metrics are retained even if the Prometheus pod is restarted or rescheduled.

Grafana is also configured to use a **PV** to preserve dashboard configurations, user preferences, and custom visualization settings. Furthermore, it uses a **PV** to ensure that if changes are made to the dashboards, they will persist across restarts without having to manually reconfigure them, providing a seamless user experience.

More details about the specific configuration of the **PVs** can be found in Section 7.2.2.

6.3.2 Resource Waste, Requests and Limits

In Kubernetes, requests and limits are crucial for managing resource allocation and ensuring efficient utilization of resources per container. Requests define the minimum amount of resources a container requires to run effectively, while limits set the maximum amount of resources a container can consume. These parameters can be set in the container specifications within Kubernetes manifests and, when properly configured, help prevent resource contention, ensure fair resource distribution, and optimize overall cluster performance. Containers can be deployed without resource requests and limits if neither is specified.

For this framework, Resource Waste is defined as the proportion of unused requested resources, indicating that these resources could have been allocated more efficiently. This metric is calculated for CPU and memory and is expressed as a percentage of the total requested resources.

The formulas for calculating wasted CPU and memory are as follows:

$$\text{Wasted CPU (\%)} = \max \left(0, \frac{\text{Requested CPU} - \text{Used CPU}}{\text{Requested CPU}} \right) \times 100 \quad (6.1)$$

$$\text{Wasted Memory (\%)} = \max \left(0, \frac{\text{Requested Memory} - \text{Used Memory}}{\text{Requested Memory}} \right) \times 100 \quad (6.2)$$

The `max` function ensures that the waste percentage stays non-negative, even if the used resources exceed the requested ones, which can happen, for example, when limits are set higher than requests or not set at all.

In cases where no resource requests and limits are defined, the waste calculation defaults to zero, and the framework generates a recommendation suggesting that the given resource's requests and limits should be set, as can be seen in Listings E.16 and E.17.

Similarly, when no limits are set, containers can consume unlimited resources, posing various risks to cluster stability, performance, and cost predictability. In such cases, the framework also defaults the waste calculation to zero, and generates a recommendation suggesting that resource limits should be defined, as seen in Listings E.14 and E.15.

When limits are defined, even if requests are not, Kubernetes will assume the requested resources are equal to the limits; however, the opposite is not true: if requests are defined, limits are not automatically set. Having this in mind, these four recommendations cover all the possible combinations of unset requests and limits, ensuring that if the framework detects any of these situations, it will generate the appropriate recommendation to address them.

6.3.3 Efficiency Score Calculation

The efficiency score is a composite metric that quantifies the cost-effectiveness of resource utilization within a Kubernetes namespace. This score combines CPU and memory waste calculations from the previous Section to comprehensively evaluate resource efficiency on a scale ranging from 0 to 100, where higher values indicate better resource utilization. The efficiency score is calculated using the following formula:

$$\text{Efficiency Score} = 100 - \frac{\text{Wasted CPU (\%)} + \text{Wasted Memory (\%)}}{2} \quad (6.3)$$

This formula subtracts the average CPU and memory waste percentages from 100 to yield the efficiency score. This means that efficiency scores closer to 100 indicate low resource waste and high efficiency, while scores closer to 0 indicate significant resource waste and inefficiency. This metric provides a more macroscopic view of resource utilization, allowing less technical users to have a general idea of the efficiency of their Kubernetes namespaces.

An exception to this behaviour occurs when requests are not set, either for CPU or memory. In this case, the waste calculation defaults to zero, and the efficiency score will result in 100, indicating that there is no waste, which may not be accurate. This outlines the importance of setting resource requests and limits, as it is not only a best practice in Kubernetes environments, but also a requirement for accurate waste and efficiency score calculations. Another way of tackling this problem is by configuring LimitRanges, which are Kubernetes resources that enforce constraints on the resource requests and limits for pods in a namespace, ensuring that all pods have defined resource requests and limits, even if they are not explicitly set in the pod specifications.

Algorithm 6 provides a pseudocode representation of the efficiency score calculation algorithm, as well as the resource waste calculation, putting together the steps described above.

6.3.4 Resource Utilization Metrics

Resource utilization is critical to managing Kubernetes clusters, as it directly impacts performance, costs, and overall efficiency.

In this framework, resource utilization is calculated as a ratio of the actual resource utilization to the requested resources for both CPU and memory.

The formulas for calculating CPU and memory utilization ratios are as follows:

$$\text{CPU Utilization Ratio} = \frac{\text{Used CPU}}{\text{Requested CPU}} \quad (6.4)$$

$$\text{Memory Utilization Ratio} = \frac{\text{Used Memory}}{\text{Requested Memory}} \quad (6.5)$$

These ratios provide direct insights into how effectively the requested resources are being used and if there is room for optimization. A utilization ratio below 1 indicates that the actual usage is less than the requested resources, suggesting potential for rightsizing. On the other hand, a ratio above 1 indicates that the actual usage exceeds the requested resources, which, depending on the context, may suggest that the requests are set too low or that the workload is experiencing spikes.

This metric, similarly to the waste calculation and the efficiency score, relies on requests and limits explained in Section 6.3.2 and, despite being a useful standalone metric for assessing resource utilization, should be interpreted along with those metrics and the recommendations generated to achieve a more comprehensive understanding of the resource utilization in the given namespace.

6.3.5 Rightsizing Recommendations and Optimization Savings

The rightsizing recommendations are generated based on an analysis of resource utilization patterns, both for CPU and memory. This analysis intends to identify opportunities to optimize resource allocation and reduce costs. During this Section, the term `Resource` refers to both CPU and memory, as the same principles and formulas apply to both. Below are the key components of the rightsizing recommendations engine:

Optimization Criteria: The framework identifies over-provisioned workloads using the resource utilization ratios defined in Section 6.3.4. A 70% utilization threshold was selected based on observations made during the experimentation phase, where utilization consistently below this threshold indicated that there was over-provisioning of resources and room for optimization. Having said this, due to the nature of this threshold, it is essential to note that it should be adjusted and tuned to the specific workload and environment, as this provides a good starting point for most workloads, but may not be suitable for all scenarios. The conditions for determining whether a workload is over-provisioned both for CPU and memory are as follows:

$$\frac{\text{Actual Usage}}{\text{Resource Requests}} < 0.7 \quad (6.6)$$

When this condition is met, optimized resource requests are calculated using a 30% safety buffer to accommodate for traffic spikes and unexpected workload increases, as follows:

$$\text{Recommended Resource Requests} = \max(0.1, \text{Actual Usage} \times 1.3) \quad (6.7)$$

Savings Calculation: The financial impact of the rightsizing recommendations is quantified by calculating the potential monthly cost savings that can be achieved by implementing them. These savings are calculated based on the difference between the current resource requests and the recommended resource requests — both for CPU and memory — and multiplying

it by the hourly cost of the resources and by the number of hours in a month, which is approximately 730 hours. More specifically, the savings are calculated as follows:

$$\text{Monthly Savings} = (\text{Current Request} - \text{Recommended Request}) \times \text{Hourly Cost} \times 730 \quad (6.8)$$

To make this more meaningful, the framework can provide a threshold for the minimum savings to be considered significant, if the users want to ignore small savings that may not justify the effort of implementing the recommendations.

This savings calculation is done when the optimization criteria are met and on a per-resource basis, meaning that the savings are calculated separately for CPU and memory.

CPU Request Rightsizing: For CPU requests meeting the optimization criteria, the framework recommends reducing the CPU requests to match actual usage patterns. Monthly savings are calculated using the configured CPU hourly cost rate.

Memory Request Rightsizing: Similarly, for memory requests meeting the optimization criteria, the framework generates a recommendation to reduce the memory requests to match actual usage patterns. Monthly savings are calculated using the configured memory hourly cost rate.

Resource Limits Addition: For workloads with set requests but missing limits, the framework recommends adding appropriate limit configurations, as described in Section 6.3.2. This prevents resource contention and improves cluster stability.

Resource Limits and Requests Addition: For workloads missing both requests and limits, recommendations are also generated. Since when limits are set, kubernetes assumes that the requests are the same, these recommendations and the above cover all possible scenarios where requests and limits are not set, as referenced in Section 6.3.2. This ensures that a user of the framework receives actionable insights to determine how to optimize their Kubernetes workloads.

These recommendations are generated by the FinOps API component, which implements the logic described above and are then scraped by Prometheus. Afterwards, the respective alerts are generated and sent to AlertManager, which notifies users through their preferred channels, such as email, or Slack.

6.3.6 Anomaly Detection Methodology

The anomaly detection algorithm uses statistical methods to distinguish between normal cost fluctuations and genuine anomalies that require attention. The algorithm establishes baseline cost patterns from historical data and quantifies deviations from these using z-scores.

The foundation of this approach rests on collecting a minimum of $n \geq 24$ hours of historical cost data to ensure statistical validity, according to the Law of Large Numbers [53], which states

that as the number of observations increases, the sample mean will converge to the expected value. This threshold ensures enough data points to calculate meaningful statistics, such as the mean and standard deviation, which are essential for calculating the z-score. This 24-hour threshold was chosen to account for daily operational cycles and fluctuations in Kubernetes workloads.

The mathematical concept behind the anomaly detection algorithm is based on the z-score analysis, which measures how many standard deviations σ the current cost $C_{current}$ deviates from the historical mean $\mu_{historical}$. The standardized Z-score is calculated as follows:

$$Z = \frac{C_{current} - \mu_{historical}}{\sigma_{historical}} \quad (6.9)$$

This formula returns a value with no units that remains consistent across namespaces and spending levels, enabling anomaly detection criteria whether analyzing a namespace's cost or the cluster's total cost.

The framework converts Z-scores into intuitive anomaly scores through a piecewise linear transformation $f : \mathbb{R} \rightarrow [0, 100]$ defined as follows:

$$S_{anomaly} = \begin{cases} 0 & \text{if } |Z| \leq 1 \\ 50(|Z| - 1) & \text{if } 1 < |Z| < 3 \\ 100 & \text{if } |Z| \geq 3 \end{cases} \quad (6.10)$$

This transformation maps Z-scores to a meaningful range of anomaly scores between 0 and 100, based on the severity of the deviation from the historical mean. It is based on the 68-95-99.7 rule, which states that for a normal distribution, approximately 68% of the data falls within one standard deviation from the mean, 95% within two standard deviations, and 99.7% within three standard deviations — providing a statistical foundation for the thresholds used in the transformation. [46]

Z-scores within the interval $(-\infty, 1]$ indicate normal fluctuations, receiving an anomaly score of 0 to indicate acceptable cost variation.

Z-scores within the interval $1 < |Z| < 3$ represent increasingly significant cost deviations, with anomaly scores scaling linearly from 0 to 100. At the threshold of $|Z| = 2$, representing the 95%

Z-scores within the interval $[3, +\infty)$ represent significant deviations beyond the 99.7% confidence level, receiving the maximum anomaly score of 100 to indicate a severe anomaly requiring immediate attention.

This algorithm also addresses the case where the standard deviation $\sigma_{historical} = 0$, which can occur when all historical costs are identical — indicating no cost variability. In this scenario, the algorithm applies the following logic:

$$S_{anomaly} = \begin{cases} 0 & \text{if } C_{current} = \mu_{historical} \\ 100 & \text{if } C_{current} \neq \mu_{historical} \end{cases} \quad (6.11)$$

This approach ensures that any deviation from the historical mean results in the maximum anomaly score of 100. This is because if the standard deviation is zero, the historical costs have not varied, and any change in the current cost is considered an anomaly.

The relative increase percentage is also calculated to provide additional financial context to the anomaly score. This percentage is calculated as follows:

$$\Delta_{relative} = \frac{C_{current} - \mu_{historical}}{\mu_{historical}} \times 100 \quad (6.12)$$

This metric quantifies the relative magnitude of cost deviations, providing a percentage increase or decrease compared to the historical mean. This helps users of the framework understand the financial impact of the anomaly when choosing how to address it.

Algorithm 5 provides a pseudocode representation of the anomaly detection algorithm, putting together the steps described above.

6.3.7 Cost Forecasting

The cost forecasting is done directly in Grafana using metrics retrieved from Prometheus. This cost forecasting is done based on cost data collected by OpenCost, which is then stored in Prometheus, and one usage metric from kube-state-metrics.

The cost forecasting accounts for the following costs: (1) node costs, which are the costs associated with the nodes in the cluster, including their hourly cost and any additional costs related to their usage; (2) **PV** costs, which are the costs associated with the persistent volumes in the cluster — **EBS** in the case of **AWS** — including their hourly cost and storage capacity; (3) load balancer costs, which are the costs associated with the external load balancers, such as **AWS** Elastic Load Balancer (**ELB**).

Listing 6.1 shows the **PromQL** query used to calculate the daily cost forecast, which sums up the hourly costs of nodes, persistent volumes, and load balancers, multiplying each of them by 24 to get the daily cost. Refer to Appendix D for more details on the metrics used in this query.

Listing 6.1: PromQL Query for daily cost forecasting

```

1 sum (
2   node_total_hourly_cost{
3     job=~"$job"
4   }
5 ) * 24
6 +
7 sum (
8   sum (
9     kube_persistentvolume_capacity_bytes{
10      job=~"$job"
11    } / (1024 * 1024 * 1024)
12  ) by (persistentvolume)

```

```

13 * on(persistentvolume) group_left()
14 sum(
15   pv_hourly_cost{
16     job=~"$job"
17   }
18 ) by (persistentvolume)
19 ) * 24
20 +
21 sum(kubecost_load_balancer_cost) * 24

```

This query can be expressed and generalized to calculate the cost forecast for any time period t in hours, as follows:

$$C_{forecast}(t) = \left(\sum_{i=1}^n C_{node_i}^{hourly} \right) \times t + \left(\sum_{j=1}^m \frac{V_j}{1024^3} \times C_{pv_j}^{hourly} \right) \times t + \left(\sum_{k=1}^l C_{lb_k}^{hourly} \right) \times t \quad (6.13)$$

Where $C_{forecast}(t)$ represents the total forecasted cost for a given time period, t is the time multiplier in hours, n is the number of nodes in the cluster, $C_{node_i}^{hourly}$ is the hourly cost of node i from the `node_total_hourly_cost` metric, m is the number of **PVs** in the cluster, V_j is the capacity of **PV** j in bytes from the `kube_persistentvolume_capacity_bytes` metric with the division by 1024^3 converting bytes to gigabytes, $C_{pv_j}^{hourly}$ is the hourly cost per GB of persistent volume j from the `pv_hourly_cost` metric, l is the number of load balancers, and $C_{lb_k}^{hourly}$ is the hourly cost of load balancer k from the `kubecost_load_balancer_cost` metric.

By default, this framework provides an hourly, daily, and monthly cost forecast where the query is essentially the same, but the time period t changes to 1, 24, or 730, respectively. These dashboards can be seen in Figure C.10.

Chapter 7

Experimental Setup and Results

This chapter presents the experimental setup used to validate the solutions proposed in this dissertation, along with the results obtained from the experiments presented in the previous chapters.

Section 7.1 describes the experimental setup and results for the CloudWatch-based Framework. Section 7.2 details the experimental setup and results for the Kubernetes-based Framework. Section 7.3 formalizes the validation methodology used to assess the effectiveness and efficiency of the proposed frameworks. Finally, Section 7.4 presents the results obtained from the experiments conducted with both frameworks.

7.1	CloudWatch-based Framework Experimental Setup	70
7.2	Kubernetes-based Framework Experimental Setup	71
7.3	Validation Methodology	74
7.4	Cost Prediction Accuracy Results	76

7.1 CloudWatch-based Framework Experimental Setup

This section describes the experimental setup used to validate the CloudWatch-based Framework and presents the results obtained from the experiments conducted with this framework.

The experimental setup for the CloudWatch-based framework was conducted using the framework’s simulator mode described in Section 5.2.3 with a production dataset from Kuehne + Nagel, so that the framework could be tested and validated without incurring more costs than the already existing ones.

The dataset used for the experiments is a production dataset from Kuehne + Nagel from two production EC2 instances of type `t3.xlarge` used as CI/CD runners. Each instance has 4 vCPUs, 16 GiB of memory, and costs 0.1664\$ per hour on AWS *us-central-1* region. The dataset was exported from Amazon CloudWatch in CSV format, containing a comprehensive set of metrics along with the timestamps for each data point, such as CPU utilization percentages, detailed network I/O for both inbound and outbound traffic, and others.

7.2 Kubernetes-based Framework Experimental Setup

This section describes the experimental setup used to validate the Kubernetes-based Framework and presents the results obtained from the experiments conducted with this framework.

During the development of this framework, `kind` [51] — a tool for running local Kubernetes clusters using Docker — was used to create a test cluster for easier and faster development and testing. At a later stage, with the goal of validating the framework in a more realistic environment, the framework was deployed on an existing **EKS** cluster on **AWS**. Both setups will be described in more detail in the following subsections.

7.2.1 Kind Cluster Configuration

While developing the framework, a local Kubernetes cluster was created using `kind` [51], which enabled us to iterate quickly and test the framework without incurring any costs. Furthermore, `kind` allowed us to develop the scripts and configurations needed to deploy the framework on a real Kubernetes cluster, such as the installation of the necessary components, the creation of namespaces, ConfigMaps, and deployments, as well as the configurations needed for each of this components to work together seamlessly; leaving us only with the task of adapting the scripts to the specific requirements of the real cluster, as described in Section 7.2.2.

The `kind` cluster was configured with a single control-plane node to keep the setup simple and lightweight while providing the essential features needed to test the framework. Additionally, the cluster port mappings were also configured so that later in the script, we could enable external access through an NGINX ingress controller, as shown in the YAML configuration below.

Listing 7.1: Kind cluster configuration

```
1 kind: Cluster
2 apiVersion: kind.x-k8s.io/v1alpha4
3 nodes:
4 - role: control-plane
5   extraPortMappings:
6   - containerPort: 80
7     hostPort: 80
8     protocol: TCP
9   - containerPort: 443
10     hostPort: 443
11     protocol: TCP
```

The configuration script can be found in Listing B.1. There are various other configurations, such as ServiceMonitors for the middleware API, Ingresses for Prometheus, Grafana, Alertmanager, and Opencost, and **PVC** configurations for simulation purposes, which are not included in this document for brevity; however, the complete configuration scripts both for the `kind` cluster and the **EKS** cluster, as well as the codebase of the framework can be found in Github [42].

7.2.2 EKS Cluster Configuration

After testing the framework on the local kind cluster, the next step involved deploying it on a real Kubernetes cluster to validate its functionalities against a more realistic environment. This transition required several adjustments to accommodate the specific requirements of the **AWS** environment. Below are the key aspects and major differences in the setup process compared to the kind cluster:

Cluster Infrastructure: The experimental setup used to validate and test this framework was conducted on an existing **EKS** cluster with two worker nodes, each running on a `m5a.xlarge` instance. The cluster at hand was already running some test workloads, such as StackRox [84], some monitoring tools, and a few other services, which, despite not being directly relevant to the framework, provided a more realistic environment for testing and validation.

Deployment Process: The deployment process was automated through a setup script that orchestrated the installation and configuration of the necessary components. This script is a direct adaptation of the one used for the kind cluster, with some slight modifications — mostly related to **AWS** specific configurations, as described below. The general overview of the deployment process is as follows: (1) creation of dedicated namespaces for the services to provide isolation and organization within the cluster; (2) installation and setup of Grafana, Opencost, Prometheus, and AlertManager through Helm charts; and (3) deployment and configuration of the custom middleware. For a more detailed perspective on the specifics of the deployment process on **AWS**, please refer to the following items and to the codebase available on Github [42].

Persistent Volumes: To ensure data persistence for both Grafana dashboards and Prometheus metrics, **PVCs** were created for each of these components. The **PVC** for Grafana was configured with a size of 10 GiB, while the one for Prometheus was set to 20 GiB. The use of **PVs** causes Helm to create StatefulSets instead of regular Deployments, ensuring that the data is not lost when the pods are restarted.

Network Access Configuration: To provide external access to the services running in the cluster, the setup script creates Ingresses for all the components. These ingresses are configured to use the NGINX ingress controller, which was already running in the cluster, to route external traffic to the correct services through LoadBalancer services, enabling users outside the cluster to access the services through the internet via **AWS** assigned hostnames.

AWS Athena Integration: During the setup of the framework, Opencost was configured to integrate with AWS Athena — an interactive serverless query service that allows users to analyze data stored in an S3 bucket using standard Structured Query Language (**SQL**) queries [45] — to retrieve cost data from the Amazon **CUR**, as can be seen in Figure 7.1. This integration enables access to dashboards and reports from **AWS** without the need to go through the **AWS** console. It also provides a user-friendly way of comparing costs across

our framework and the ones provided by **AWS** and clearly exposes the lack of real-time cost data from **AWS** reports. The integration was configured to use the Amazon **CUR** data stored in an S3 bucket, which is updated daily with the latest cost data from **AWS**.

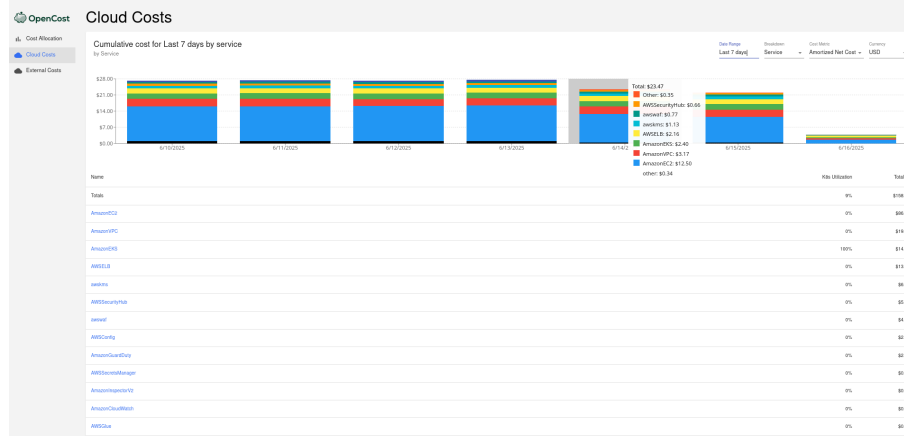


Figure 7.1: Opencost UI Integration with AWS Athena

Custom Middleware Deployment: The middleware **API** was packaged using a custom Docker image and deployed to Amazon Elastic Container Registry (**ECR**) — a fully managed container registry service [28] — to ease the deployment process. The deployment itself was configured to use environment variables for the necessary configurations, such as the addresses of Prometheus, Opencost, Grafana, and some cost data related variables. The middleware was also deployed with resource requests and limits, not only to ensure that the **API** has enough resources to run smoothly, but also to provide data for the framework to work with. At last, since this **API** is essential to calculate savings, recommendations, and other cost-related data, we configured health checks to ensure that it is operating correctly and reliably within the cluster.

Operation Constraints and Cost Calculation: Our framework assumes that the cluster is running 24 hours a day, 7 days a week, which is the default behavior of Kubernetes clusters. Under this assumption, cost calculations use 730 hours per month as the baseline, which is the average number of hours in a month, calculated as $\frac{365 \text{ days} \times 24 \text{ hours}}{12 \text{ months}} = 730 \text{ hours}$. However, in this specific scenario, the **EKS** cluster is only running on weekdays from 9 AM to 8 PM, providing 11 operational hours per day, 5 days a week. To account for this constraint, the monthly operational hours were adjusted to reflect the actual usage of the cluster. Given that there are approximately 262 working days per year [43], the adjusted number of operational hours per month will be $\frac{262 \text{ days} \times 11 \text{ hours}}{12 \text{ months}} \approx 240.167 \text{ hours}$. After having this value calculated, both the framework and the Grafana dashboards were configured to use this value instead of the default 730 hours, ensuring that the cost calculations are accurate and reflect the actual usage of the cluster.

These configurations and adjustments were essential to ensure that the framework operates correctly in the **AWS** environment and provides accurate cost predictions and insights. The deployment process was automated through a setup script. A version for KIND can be found in Appendix **B.1.2**.

7.3 Validation Methodology

This section focuses on the validation methodology of the frameworks developed during this dissertation.

The validation methodology for both frameworks is structured around two primary factors: (1) Cost prediction accuracy, which assesses the precision of real-time cost estimates generated by the frameworks; and (2) User demand evaluation, which evaluates the relevance of the frameworks and their features to potential users.

7.3.1 Cost Prediction Accuracy Criteria

The cost prediction accuracy of the Kubernetes-based framework is quantitatively assessed using Mean Absolute Percentage Error (**MAPE**), a widely adopted metric in forecasting literature that quantifies the average percentage error between actual and forecasted values [3, 38, 54].

MAPE is defined as the mean absolute percentage error between the actual and forecasted values, providing an intuitive interpretation of prediction accuracy, with lower **MAPE** values indicating superior forecasting performance. **MAPE** is a normalized measure of prediction accuracy that is independent of the scale of the data, making it suitable for this research. To calculate **MAPE**, the following formula is used:

$$\text{MAPE} = \frac{1}{n} \sum_{i=1}^n \left| \frac{A_i - F_i}{A_i} \right| \times 100 \quad (7.1)$$

where A_i represents the actual cost values, F_i represents the forecasted cost values, and n is the number of observations.

FinOps Foundation defined a cloud forecasting maturity-based matrix that reflects the maturity of cloud forecasting practices across organizations per forecasting area, such as tooling, variance, and granularity. Organizations operating at the Run maturity level — characterized by advanced automation, comprehensive tagging strategies, and sophisticated monitoring systems — target 5% forecast-to-spend variance or better. Walk maturity organizations, with developing processes and partial automation, maintain variance within 10%. Crawl maturity organizations, representing early-stage cost management implementations, accept up to 20% variance while building foundational capabilities [38].

Lewis (1982) [54] also provides an interpretation of typical **MAPE** values for forecasting, which can be summarized as follows:

For the CloudWatch-based framework, this metric is not applicable, as the framework's cost predictions are based on the actual costs of the **EC2** instances used, which are known and do not

Table 7.1: Interpretation of typical MAPE values

MAPE	Interpretation
< 10	Highly accurate forecasting
10–20	Good forecasting
20–50	Reasonable forecasting
> 50	Inaccurate forecasting

Source: Lewis (1982) [54]

require forecasting techniques. Instead, we are going to do a sanity check of the values calculated by the framework against AWS Pricing Calculator [11].

For this dissertation, we establish a 5% threshold for **MAPE** for cost prediction accuracy validation, positioning our evaluation among the Run maturity level of the FinOps Foundation’s cloud forecasting maturity matrix and as a highly accurate forecasting practice according to Lewis (1982) [38, 54]. The 10% threshold ensures that cost predictions provide sufficient reliability for strategic business decision-making while acknowledging the inherent variability in cloud computing environments, workload patterns, and others [26].

7.3.2 User Demand Evaluation Criteria

The user demand evaluation will be conducted through a survey designed to gather feedback from potential users. The survey (refer to Appendix F) is structured to collect insights on the relevance, desirability, and perceived value of the proposed frameworks’ features. By analyzing the responses, the evaluation will determine whether the functionalities align with user needs and expectations, providing insights into their potential adoption and overall impact. This approach ensures a more user-focused manner of validating the system’s demand and utility. The survey will include questions about various aspects, such as:

User Background: What is your primary role? What is the approximate size of your organization? Have you ever worked with any cloud platform? Have you ever managed or deployed applications on a Kubernetes environment? To what extent do cloud costs affect your decisions?

Current Cloud Cost Management: How would you rate your current cloud cost visibility? What cloud cost challenges has your team faced? How much time does your team currently spend on manual cloud cost analysis per month? What is your current process for identifying cost optimization opportunities? How do you currently monitor Kubernetes costs? What are your biggest pain points with the current Kubernetes cost management?

Technical requirements and features: How would you rate the features of this framework? Which cost optimization alerts would be most valuable? What cost visibility granularity is most important to you?

Implementation and Adoption: How important is easy deployment for adopting a new monitoring solution? What would be the biggest barriers to adopting these frameworks? What would success look like for these frameworks? How much cost reduction would justify implementing these frameworks? Beyond cost savings, what other benefits would be valuable? Based on the description, how likely would you be to use these frameworks?

The survey was sent to potential users of the frameworks, such as cloud architects, DevOps engineers, system administrators, and other professionals with Information Technology (IT) and cloud computing background.

An initial analysis of the survey results was conducted in Chapter 4 to assess the relevance and demand for the proposed solution and to identify pain points and challenges faced by potential users in cloud cost management. These results were used to guide the development of the frameworks and to ensure that they address the needs and expectations of the target audience.

This second analysis will be conducted for validation purposes, and, since the frameworks were already developed and its feature set is already defined, it will be from a more technical perspective, focusing not only on the current challenges faced by potential users, but also on the feature set of the frameworks and how they align with the needs and expectations of the target audience.

7.4 Cost Prediction Accuracy Results

This section presents the calculations and presents the cost-related results obtained from the experiments conducted with both frameworks, as described in Sections 7.1 and 7.2. The techniques and criteria used to validate these frameworks are described in Section 7.3.

7.4.1 Kubernetes-based Framework Cost Prediction

After deploying the Kubernetes-based framework on the EKS cluster, as described in Section 7.2.2 and collecting 20 days of data — excluding weekends, as described in Section 7.2.2 — the following results were obtained:

Table 7.2: Kubernetes-based Framework's experiments cost breakdown

Date	EC2-Instances (\$)	EBS (\$)	ELB (\$)	Actual Value (\$)	Forecasted Value (\$)
2/06/2025	3.87	0.56	0.54	4.97	4.94
3/06/2025	3.87	0.56	0.54	4.97	4.94
4/06/2025	3.83	0.56	0.54	4.93	4.89
5/06/2025	3.87	0.56	0.54	4.97	4.89
6/06/2025	3.83	0.56	0.54	4.93	4.83
9/06/2025	3.83	0.59	0.74	5.16	5.05
10/06/2025	3.83	0.59	1.08	5.50	5.43
11/06/2025	3.83	0.59	1.08	5.50	5.43
12/06/2025	3.83	0.59	1.08	5.50	5.43
13/06/2025	3.87	0.56	1.08	5.51	5.43
16/06/2025	3.83	0.56	0.77	5.16	5.10
17/06/2025	3.83	0.56	0.54	4.93	4.89
18/06/2025	3.83	0.56	0.54	4.93	4.89
19/06/2025	3.83	0.55	0.54	4.92	4.86
20/06/2025	3.83	0.55	0.54	4.92	4.83
23/06/2025	3.87	0.55	0.54	4.96	4.83
24/06/2025	3.83	1.17	0.54	5.54	5.56
25/06/2025	3.83	1.91	0.54	6.28	6.15
26/06/2025	3.83	0.13	0.54	5.50	5.34
27/06/2025	3.86	0.55	0.54	4.95	4.83

Table 7.2 showcases the daily costs predicted by the FinOps framework compared to costs retrieved from AWS Cost Explorer. The table includes the breakdown of costs in two categories, with columns representing the following:

EC2-Instances (\$): This column represents the costs directly associated with the EC2 instances running in the cluster, such as their hourly usage. These instances form the backbone of the cluster and are the primary source of its costs.

EBS (\$): This column represents the costs related to PVs, used by stateful services such as Prometheus.

ELB (\$): This column represents the costs associated with Load Balancers (LBs), including services like the NGINX ingress controller.

Actual Value (\$): This column represents the sum of the previous three columns — the total costs for EC2 instances, storage, and load balancers. These costs were retrieved from AWS Cost Explorer.

Forecasted Value (\$): This column represents the costs predicted by the FinOps framework.

With this data, the MAPE can be calculated using the formula that was introduced in Section 7.3.1.

Firstly, we can calculate the absolute percentage error for each observation. For brevity, only four observations are shown below, but the same process is applied to all observations in the dataset:

$$\begin{aligned}
 \text{APE for 2/06/2025:} & \quad \left| \frac{4.97 - 4.94}{4.97} \right| \times 100 = 0.60\% \\
 \text{APE for 3/06/2025:} & \quad \left| \frac{4.97 - 4.94}{4.97} \right| \times 100 = 0.60\% \\
 \text{APE for 4/06/2025:} & \quad \left| \frac{4.93 - 4.89}{4.93} \right| \times 100 = 0.81\% \\
 & \quad \vdots \\
 \text{APE for 27/06/2025:} & \quad \left| \frac{4.95 - 4.83}{4.95} \right| \times 100 = 2.42\%
 \end{aligned}$$

Now, these absolute percentage errors can be summed up:

$$\begin{aligned}
 \text{Total APE} &= 0.60 + 0.60 + 0.81 + 1.61 + 2.03 + 2.13 + 1.27 + 1.27 + 1.27 + 1.45 \\
 &+ 1.16 + 0.81 + 0.81 + 1.22 + 1.83 + 2.62 + 0.36 + 2.07 + 2.91 + 2.42 \\
 &= 29.25\%
 \end{aligned}$$

Finally, we can calculate the **MAPE** by dividing the total absolute percentage error by the number of observations, which is 10 in this case:

$$\text{MAPE} = \frac{29.25}{20} = 1.46\% \quad (7.2)$$

With a **MAPE** of 1.46% for these experiments, the framework's cost prediction accuracy is well within the 5% threshold established for this research, indicating a highly accurate forecasting performance according to both FinOps Foundation's cloud forecasting maturity matrix and Lewis (1982) [38, 54].

7.4.2 CloudWatch-based Framework Sanity Check

As mentioned in Section 7.3.1, the CloudWatch-based framework's cost predictions are based on the actual costs of the **EC2** instances used, not on cost forecasting techniques. Instead, we performed a sanity check of the values calculated by the framework against AWS Pricing Calculator [11].

For this, we started by running the AWS Pricing Calculator for the two **EC2** instances used in the experiments with the following parameters:

Instance Type: `t3.xlarge` (4 vCPUs, 16 GiB of memory);

Region: *us-central-1* (Frankfurt, Germany);

Operating System: Linux;

Tenancy: Shared Instances;

Workloads: Constant usage;

Payment options: On-Demand;

Number of instances: 2.

This configuration resulted in a total cost of 280.32\$ per month, which matches exactly the cost calculated by the framework, which is as follows:

$$\text{Monthly Cost} = 0.192(\text{cost/hour}) \times 730 (\text{avg hours/month}) \times 2 (\text{instances}) = 280.32\$ \quad (7.3)$$

This confirms that the framework's cost calculations are accurate and consistent with the actual costs of the **EC2** instances used in the experiments.

Chapter 8

Conclusions

This chapter analyzes the results shown in the previous chapter, providing insights into the performance and effectiveness of the developed solutions. It also answers the research questions formulated in Section 4.3 and discusses the implications of the findings. Furthermore, it synthesizes this research’s key findings and contributions while addressing the limitations and challenges faced and outlines potential directions for future work based on the insights gained throughout this dissertation.

Section 8.1 answers the research questions, providing a brief overview of the findings and insights gained during this work. Section 8.2 revisits the hypothesis formulated in Section 4.2 and discusses whether it was confirmed. Section 8.3 discusses the results obtained both for the CloudWatch-based Framework sanity check and the Kubernetes-based Framework cost prediction accuracy. It also analyses the user demand survey results, providing insights into the practical relevance and market applicability of the frameworks developed in this dissertation. Section 8.4 addresses the threats to validity of the findings presented in this dissertation, discussing the limitations and potential biases that may affect the generalizability of the results. Section 8.5 summarizes the main contributions made throughout this dissertation. Section 8.6 discusses the limitations and challenges faced during this dissertation. Finally, Section 8.7 outlines potential directions for further research and development on this topic.

8.1	Research Questions	81
8.2	Hypothesis Revisited	83
8.3	Thesis Validation	83
8.4	Threats to Validity	87
8.5	Summary of Contributions	88
8.6	Limitations and Challenges	89
8.7	Future Work	89

8.1 Research Questions

For that, the research questions are addressed below, and a brief answer is provided based on the findings during this work:

RQ1: Which tools can be used to retrieve data from cloud environments in real time? Do they require any licensing? Do they support various **CSPs**? Do they incur additional costs?

For real-time data retrieval, the frameworks developed in this dissertation leverage existing tools, namely Amazon CloudWatch and OpenCost. For the case of CloudWatch, it is an **AWS** only service, and despite having a free tier, it incurs additional costs depending on the number of metrics and logs collected.

OpenCost, on the other hand, is an open-source tool that can be used in any Kubernetes cluster, regardless of the flavor or underlying **CSP**, if any is used. Since it is open-source, it does not require any licensing and is free to use.

RQ2: Which algorithms and heuristics should be developed to catalog cloud resources, monitor costs, and optimize resource allocation? Are there other metrics that should be taken into account? Do they need historical cloud cost data?

The algorithms and heuristics developed in this dissertation focus on real-time resource and cost monitoring, cataloging cloud resources, and providing cost optimization recommendations. The algorithms are based on the current resource utilization patterns and metrics retrieved from the respective environments — CloudWatch or the Kubernetes cluster. These algorithms are thoroughly described in Sections 5.3.4 and 6.3.

RQ3: Which actions or policies can be recommended to reduce cloud costs, improve resource efficiency, and enable more informed decision-making? How can policies be tailored to different service cost models? How could forecasting based on previous usage affect this? To what extent can this be automated?

The actions and policies recommended by the frameworks developed in this dissertation are based on the current resource utilization patterns and custom metrics or real-time metrics retrieved from CloudWatch, Opencost, and other tools used by the frameworks. These recommendations were based on the insights gathered from the User Demand Survey Analysis presented in Sections 4.6.1 and 8.3.3, which not only provided direct user feedback on the most desirable alerts and recommendations, but also showcased the most common challenges and pain points faced by users when managing their cloud costs and resources. This knowledge was then used to craft the most relevant recommendations for each framework, which are presented in Appendix E along with detailed explanations for each recommendation. In each framework, these recommendations are tailored to the specifics of the target at hand, such as **EC2** instances or Kubernetes clusters. They are also easy to understand and can be applied by users with different levels of expertise, containing detailed descriptions of the problem and, when possible, which actions should be taken to solve it. When

applicable, the frameworks also provide metadata about the recommended actions, such as the estimated cost savings, so that users can decide whether it is worth applying the recommendations. These alerts, along with explanations on each of them, can be found in Appendix E.

RQ4: How do real-time cost monitoring and action recommendations improve a tenant's financial operations compared to cost reports? What are the benefits regarding cost savings and operational expenses?

Real-time cost monitoring and action recommendations provide several advantages over traditional cost reports. Firstly, they enable proactive cost management by identifying and addressing inefficiencies as they occur rather than after the fact. This can lead to significant cost savings by preventing overspending before it happens. Secondly, real-time insights allow for faster decision-making, as users of the frameworks can quickly adapt to changing usage patterns and optimize resource allocation on the fly. Finally, the automation of recommendations reduces the manual effort required to analyze costs and implement changes, freeing up valuable time for teams to focus on higher-level strategic initiatives.

RQ5: How can the system be integrated into existing cloud environments? What are the requirements for integration? How can it be made compatible with existing tools and workflows?

Both frameworks developed in this dissertation were designed to be easily integrated into existing cloud environments and workflows by leveraging industry-standard tools such as Prometheus and Grafana for monitoring and visualization, as well as for providing REST APIs for custom-developed APIs within the frameworks. Both frameworks can be deployed in both local development environments, such as LocalStack and Kind, and production environments within the cloud, such as AWS for the CloudWatch-based framework and EKS clusters for the Kubernetes-based framework. The frameworks come bundled with pre-configured dashboards and alerts without the need for additional configurations, making them easy to use and understand by users with different levels of expertise. Having said this, the frameworks can be easily tuned to specific environments by modifying the configuration files, which are well documented within the codebase. Furthermore, the frameworks come with scripts that automate the deployment process, making it so that user can deploy and integrate them into their existing environments without the need to understand the underlying implementation details of the frameworks. The script for local deployment on Kind of the Kubernetes-based framework is available in Appendix B.1.2 for reference.

This work has successfully addressed the research questions, providing an overview of the tools, algorithms, and heuristics used to monitor and optimize cloud costs in real-time. The next chapter revisits the central hypothesis of this dissertation, evaluating whether it was confirmed based on the findings gathered throughout this work.

8.2 Hypothesis Revisited

The central hypothesis of this dissertation was formulated as follows:

H: *A system that monitors and forecasts costs in real-time can provide decision-makers with indicators that lead to measurable cost savings and better resource allocation.*

The findings gathered by answering the research questions that guided this dissertation help to evaluate the hypothesis, which can be summarized as follows:

A system that monitors and forecasts costs in real time... Both frameworks developed during this dissertation are capable of monitoring and forecasting costs in real-time, as demonstrated by the analysis presented in Section 8.3.1 and 8.3.2. While the first framework's focus is not on forecasting costs, it can provide real-time cost calculations based on pre-configured values and metrics. Furthermore, it can perform real-time resource analysis, directly impacting costs, as described in Section 2.1.1, providing users with another way to gain real-time indirect insights into their costs. The second framework is capable of providing real-time, detailed cost predictions and forecasts for Kubernetes clusters, as well as for additional services such as ELB and EBS, as shown in Section 7.4.1.

...can provide decision-makers with indicators that lead to measurable cost savings and better resource allocation. By providing real-time cost and resource visibility, as well as cost predictions and forecasts, the frameworks developed in this dissertation enable decision-makers to make informed decisions about their cloud resources, leading to measurable cost savings and better resource allocation. Furthermore, the frameworks provide detailed recommendations (refer to Appendix E) for cost and resource optimization. These recommendations provide detailed explanations of the problems and explicit cost savings estimates, when applicable. With these real-time observability and detailed recommendations, decision-makers can take proactive actions to optimize their cloud costs and resources, leading to better resource allocation and measurable cost savings.

In conclusion, the central hypothesis of this dissertation is confirmed, showing that *a system that monitors and forecasts costs in real-time can provide decision-makers with indicators that lead to measurable cost savings and better resource allocation.*

8.3 Thesis Validation

This section analyzes and gives a final assessment of the cost prediction accuracy results presented in Section 7.4.1 and 7.4.2.

8.3.1 Kubernetes-based Framework Cost Prediction

The cost prediction accuracy results for the Kubernetes-based framework, presented in Section 7.4.1, showcase a MAPE of 1.46%, indicating a high level of accuracy in the cost predictions made by

the framework. According to the criteria defined in Section 8.3, this **MAPE** value is well below the defined validation threshold of 5%, which confirms that the framework is capable of providing highly accurate real-time cost predictions for Kubernetes clusters. Having said this, and considering the results presented in Table 7.2, the differences between the values predicted by the framework and the actual costs incurred in the cluster — despite being very small, and mostly negligible — are partly due to the fact that, in the experimental setup, EC2 cluster nodes take between 10 and 20 minutes to fully shut down. As a result, the cost associated with these nodes does not correspond to exactly 11 hours of usage, as estimated in Section 7.2.2, but rather to 11 hours plus an additional 10 to 20 minutes.

Nevertheless, the results successfully demonstrate that daily costs vary consistently in real-time as changes are made. The following examples present on Table 7.2 illustrate this:

Persistent Volume addition: On June 9 a 10GB **PV** was added and, on June 24 a 500GB **PV** was added. This resulted in immediate cost increases proportional to the size of the added volumes.

Persistent Volume removal: On June 13 a 10GB **PV** was removed and, on June 19 two **PVs** totaling 10GB were removed. This resulted in immediate cost decreases proportional to the size of the added volumes.

Load Balancer addition: On June 9, a LoadBalancer-type service was deployed, leading to an increase in both **ELB** and overall daily costs.

Load Balancer removal: On June 16, a LoadBalancer-type service was deleted, resulting in a corresponding reduction in **ELB** and daily costs.

Summing up, the results confirm that this framework is capable of providing highly accurate real-time cost predictions for Kubernetes clusters, improving the visibility of cloud costs and resources compared to native solutions such as **AWS** Cost Explorer, which provided this data only the day after the costs were incurred.

8.3.2 CloudWatch-based Framework Sanity Check

The Sanity Check results, presented in Section 7.4.2, confirm that the framework's cost calculations are accurate and consistent with the actual base costs of the **EC2** instances used in the experiments. The results show that the framework correctly calculates the base costs of the **EC2** instances, as the calculated costs match the expected costs based on the instance type and region. Despite being a simple test and not the principal focus of the CloudWatch-based framework, it shows improvements regarding native **AWS** tools, such as the **AWS** Cost Calculator, which does not provide real-time cost calculations and requires manual input of the instance type and region.

8.3.3 Reviewing user demands taking into account technical details

The complete survey results, including detailed demographic breakdowns and feature evaluations, are presented in Appendix F.2.

The user demand survey results provide user-backed evidence for the practical relevance and market applicability of automated cloud cost optimization frameworks. This section analyzes the survey results separated by key themes, highlighting the insights gathered from the responses and their implications for the frameworks developed in this dissertation.

Respondent Profile and Representativeness: The survey demographics reveal a highly qualified and experienced respondent pool, which strengthens the validity of the findings. With 85.7% of respondents working in large enterprises and 90.5% currently working with cloud platforms, the survey effectively captures the perspectives of organizations with substantial cloud operations. The role distribution of respondents is dominated by 38.1% of DevOps/Platform Engineers, 28.6% of Software Developers, 19% of Engineering Managers or Team Leads, 9.5% of Site Reliability Engineers, and 1% of FinOps Analysts, indicating a high proportion of technical professionals who are usually involved in cloud operations and decision-making.

Current State Analysis: The survey results indicate odd findings regarding the current state of cloud cost management practices. While 76.2% of respondents indicate that cloud costs play a significant role in decision-making, only 23.8% report having comprehensive visibility and control over cloud costs. This gap between importance and capability represents a clear market opportunity for the frameworks developed in this dissertation, as they enable users to gain real-time visibility and control over their cloud costs.

The time investment patterns also highlight this gap, with 85.7% of respondents spending between 1 and 30 hours monthly on manual cost analysis. This suggests that organizations still dedicate significant time and resources to manual processes for cloud cost analysis, leading to Operational Expenses (OPEX) inefficiencies in activities that could be partially automated. The prevalence of manual cost analysis is further supported by an overwhelming 90.5% of respondents indicating that they rely on AWS Cost Explorer for cost optimization, 38.1% on periodical monthly or quarterly reports, and only 28.6% on custom or third-party tooling. These findings support the idea that native tools such as AWS Cost Explorer are not sufficient for the complexity and scale of modern cloud environments. The combination of high manual effort and limited visibility creates an environment where cost optimization opportunities are frequently missed.

Challenges and Pain Points: The challenges that respondents identified in the survey directly align with the design decisions made in both frameworks. 81% of respondents indicated already having experienced unexpected cost spikes, justifying the frameworks' focus on anomaly detection and real-time alerting capabilities. Furthermore, 71.4% of respondents

reported that they struggle with the manual effort required for cost analysis, while 66.7% indicated having over-provisioned resources. These findings confirm the need for automated optimization recommendations and efficiency scoring mechanisms, which are key features of both frameworks.

For Kubernetes environments specifically, 77.8% of respondents reported a lack of granular visibility and real-time insights over costs, which aligns with the Kubernetes-based framework's focus on providing real-time detailed cost visibility at multiple levels of granularity, such as cluster-level, namespace-level, node-level, and pod-level. The survey also revealed the preference for cluster-level and namespace-level visibility, with 72.2% and 66.7% of respondents, respectively, indicating these as their preferred levels of granularity.

Feature Evaluation and Value Proposition: The feature evaluation results provide strong user validation for the frameworks' core functionalities. For the CloudWatch-based framework, the highest-rated feature — unexpected cost spikes and budget overruns prevention — received a mean score of 4.91 out of 5, showcasing the critical need for proactive cost management. The second and third highest-rated features — automated alerts for optimization opportunities and real-time efficiency scoring and recommendations — confirm that automation is a key feature for potential users of the framework, with mean scores of 4.57 and 4.48, respectively.

Similarly, the Kubernetes framework's cost anomaly detection features and alerts received the highest ratings, with a mean score of 4.78 out of 5, followed by real-time dashboards per cluster, namespace, node, and pod, with a mean score of 4.67. The similarly high rating of 4.44 for Grafana and Prometheus integration further supports the frameworks' design decisions to leverage these widely adopted tools for monitoring and visualization.

Interestingly, the relatively lower ratings of 3.61 for deployment ease features — one-click deployment on existing workflows for the Kubernetes-based framework — suggest that users prioritize functionality over convenience. Similarly, the CloudWatch-based framework's multiple operation modes received a mean score of 3.43, indicating that while users value flexibility, it is not as critical as the core functionalities of the frameworks.

Implementation Barriers and Success Criteria: The identification of security and compliance concerns as the primary implementation barrier for these frameworks by 61.1% of respondents highlights a critical consideration not extensively addressed in the framework development. Despite that, the frameworks' reliance on widely adopted, tested, and patched tools such as Prometheus, Grafana, OpenCost, and CloudWatch mitigates many of these concerns, as these tools are designed with at least some degree of security in mind. Having said this, the frameworks can be further improved in future iterations to address these concerns more thoroughly, such as by implementing role-based access control, encryption, and other security best practices.

Additionally, the 44.4% of respondents indicating that resource overhead on clusters would be a barrier to implementation validate the frameworks' design approaches, particularly the resource efficiency focus of the Kubernetes-based framework.

The success criteria emphasize overall cost reduction and faster cost optimization identification, with 83.3% and 61.9%, respectively, aligning well with the frameworks' capabilities too.

Adoption Likelihood and Market Potential: The survey's most encouraging finding is the unanimous positive adoption likelihood, with 72.2% of respondents indicating they would be very likely to adopt the frameworks and 27.8% indicating they would be likely to do so. This 100% positive response rate suggests a strong potential for these frameworks' adoption. While potentially biased by the survey's target audience being cost-aware users, it is still a strong indicator of the framework's potential. For the most important benefits of implementing these frameworks beyond cost savings, the survey results show that the two most valued benefits are improved cost awareness and visibility and improved resource utilization efficiency, with 94.4% and 83.3% respondents' votes, respectively, suggesting that the users recognize the frameworks' values extend beyond direct financial savings.

With these insights, the user demand survey results provide strong evidence for the practical relevance and market applicability of the frameworks developed in this dissertation.

8.4 Threats to Validity

Several factors may limit the generalizability and validity of the findings presented in this dissertation.

The User Demand Survey, while providing valuable insights into user needs and preferences, is based on a relatively small sample of 21 respondents, which may not fully represent the broader population of cloud tenants and IT professionals. Additionally, even though the survey was distributed to several companies' IT professionals, such as DevOps Engineers, Software Developers, and Engineering Managers, the sample may still be biased towards users who are already familiar with FinOps and cloud cost management practices due to the interest of the respondents in the topic, potentially skewing the results towards those who are more likely to adopt such frameworks.

The cost prediction validation experiments were conducted over a limited time period and within a specific controlled environment, primarily using Amazon EKS clusters and EC2 instances. This validation methodology relied on semi-controlled scenarios, which, while effective for demonstrating the frameworks' capabilities, may not fully capture the complexities and variations of real-world cloud environments and the unpredictability of cloud costs. The evaluation period, while sufficient for demonstrating basic functionality, may be too short to capture longer-term trends, seasonal variations, or edge cases that could affect cost prediction accuracy and optimization recommendations. These limitations suggest that while the frameworks show promising results in these environments, further validation in diverse and larger-scale production

environments and extended time periods would strengthen the confidence in their practical applicability and effectiveness.

8.5 Summary of Contributions

This dissertation has made significant contributions to the field of **FinOps** by developing two novel frameworks for real-time resource and cost monitoring in cloud environments.

The first framework, which is built on top of Amazon CloudWatch, provides a comprehensive solution for monitoring cloud resources in real-time. While the focus of this framework is not on cost forecasting, it does provide a basic cost prediction model based on instance pricing specifications and monthly calculations. The efficiency scoring algorithms and optimization recommendations are derived from current resource utilization patterns — CPU, memory, and network usage — enabling automated detection of overprovisioned and underutilized resources. The framework integrates CloudWatch metrics with Prometheus for time-series storage and Grafana for visualization, supporting multiple operational modes, including LocalStack-based development environments and production AWS deployments. A key strength of this framework lies in its extensible and modular architecture, which promotes future extension to additional AWS services beyond **EC2**, requiring only the implementation of service-specific connectors and corresponding analyzers while reusing the core components without the need to understand the underlying implementation details of the framework.

The second framework, which is built on top of OpenCost, provides comprehensive cost monitoring and optimization for Kubernetes clusters. This framework monitors costs across namespaces, pods, containers, **PVs**, and load balancers, generating cost efficiency scores and identifying underutilized resources. This system can be deployed in various environments, including local development setups — such as Minikube and Kind — and production Kubernetes clusters, with one of its features being the ease of deployment with a script that automatically deploys the framework in a Kubernetes cluster¹. This framework provides automated rightsizing recommendations for CPU and memory resources, detects cost changes, and generates hourly, daily, and monthly cost forecasts. This solution's primary strength lies in its ability to provide granular cost breakdowns and real-time observability over costs, resource utilization, and storage, which are essential for effective cost management and optimization in Kubernetes environments. Similarly to the **EC2** framework, this framework also uses industry-standard tools such as Prometheus and Grafana. It provides all of its functionalities through a set of pre-configured dashboards, which can be easily understood and used by users with different levels of expertise.

Both frameworks were developed with the same goal in mind — to provide real-time observability over cloud resources and costs — and, despite their differences, they share various architectural decisions. These frameworks, used together, offer granular real-time visibility over resources and expenses, enabling cloud tenants to make informed decisions and prevent unexpected costs.

¹Currently, the script is only available for Kind and **EKS** clusters, however, it can be easily adapted to other Kubernetes flavors.

8.6 Limitations and Challenges

During the initial stages of this research, it became evident that the existing tools related to real-time cost monitoring and prediction were limited in their capabilities, and most of them were not open source. This, and the lack of focused research, made it necessary to develop the first framework from scratch, which posed a significant challenge.

For instance, the initial framework was developed based on [EC2](#), for which some literature was available; however, during the development stage, it became clear that most of this literature focused on simple proof-of-concept implementations rather than comprehensive solutions. This service, despite being seen as the most fundamental service in [AWS](#), has a very complex pricing model, with prices depending on various factors such as instance type, region, usage patterns, network traffic, billing model — on-demand, reserved, spot instance, and others — storage configuration, and, possibly additional services. This complexity made it very difficult to consider all the factors that influence the cost of a given instance. Having said this, due to the complexity of this cost model and the time and resource constraints posed during this research, the framework's cost prediction model is limited to the hourly cost of the instance and to the region in which it is deployed. Even though this is not the framework's focus, it is still a limitation that should be addressed in future work.

As for the second part of this dissertation, the Kubernetes-based framework, the main challenge was to learn and understand both Kubernetes and OpenCost — which are extensive and somewhat complex technologies — in a short period of time. Understanding the architecture of Kubernetes and its intricacies was crucial for the development of the framework, and therefore, it was necessary to invest a significant amount of time in learning and experimenting with these technologies.

8.7 Future Work

The frameworks developed during this dissertation provide a solid and novel foundation for real-time resource and cost monitoring in cloud environments. However, some areas can be extended in the future to enhance their capabilities, integrability, and usability.

Starting with the CloudWatch-based framework, as mentioned in [Section 8.6](#), the cost predictions could be improved by taking into account more factors that influence the cost of a given instance, such as [EBS](#) storage, network traffic, and different billing models. Additionally, the framework could be extended with a module that automatically scrapes the most recent information about the instances and their costs from [AWS](#) and populates the configuration files accordingly. This would remove the need for manual configuration and allow the framework to adapt to changes in the cloud environment more easily.

Regarding the Kubernetes-based framework, the costs are calculated based on the values retrieved from OpenCost, and enhanced metrics are provided through custom middleware. Since Opencost removed a lot of the complexity of the cost calculation, such as retrieving the prices

from **AWS** depending on the region and various other factors, this framework was able to provide a more accurate cost prediction model and a production-ready framework, as we could observe in the previous chapters. However, there are still some areas that can be improved in the future, such as integrating networking costs into the cost calculation, which is currently not taken into account and, depending on the use case, might significantly impact the overall cost of the Kubernetes cluster. On top of that, the framework's potential savings calculation could be extended to account for **PVs** and **LBs**, which are not used or necessary.

On top of that, these frameworks were developed with different codebases due to the different technologies and architectures used in each of them. Having said that, there is the potential to merge these two frameworks into one, enabling a more unified approach to real-time resource and cost monitoring across different environments. By doing this, we could leverage the strengths of both frameworks. Furthermore, since both frameworks have a middleware **API** that ingests data from the respective data sources and uses it to create custom metrics, it would be useful to leverage this on a larger scale to simplify **PromQL** queries in Grafana dashboards and alerts.

References

- [1] International Energy Agency. *Electricity 2024*. IEA, Paris, 2024. URL: <https://www.iea.org/reports/electricity-2024/>.
- [2] International Energy Agency. *Electricity 2024 - Executive Summary*. IEA, Paris, 2024. URL: <https://www.iea.org/reports/electricity-2024/executive-summary/>.
- [3] Arize AI. *Mean Absolute Percentage Error (MAPE): What You Need to Know*. 2024. URL: <https://arize.com/blog-course/mean-absolute-percentage-error-mape-what-you-need-to-know/>.
- [4] Alertmanager | Prometheus. URL: <https://prometheus.io/docs/alerting/latest/alertmanager/> (visited on 06/05/2025).
- [5] Amazon EC2 - Cloud Compute Capacity - AWS. Amazon Web Services, Inc. URL: <https://aws.amazon.com/ec2/> (visited on 06/28/2025).
- [6] Amazon EC2 Spot Instances - Product Details. Amazon Web Services, Inc. URL: <https://aws.amazon.com/ec2/spot/pricing/> (visited on 06/28/2025).
- [7] Amazon EKS Hybrid Nodes Overview - Amazon EKS. URL: <https://docs.aws.amazon.com/eks/latest/userguide/hybrid-nodes-overview.html> (visited on 06/28/2025).
- [8] Amazon Web Services. *EC2 Instance Metrics - AWS Compute Optimizer*. 2025. URL: <https://docs.aws.amazon.com/compute-optimizer/latest/ug/ec2-metrics-analyzed.html>.
- [9] APM Tool - Amazon CloudWatch - AWS. URL: <https://aws.amazon.com/cloudwatch/> (visited on 06/05/2025).
- [10] Asyncio — Asynchronous I/O. URL: <https://docs.python.org/3/library/asyncio.html> (visited on 06/20/2025).
- [11] AWS Pricing Calculator. URL: <https://calculator.aws/#/> (visited on 06/23/2025).
- [12] AWS SDK for Python - Boto3 - AWS. URL: <https://aws.amazon.com/sdk-for-python/> (visited on 06/20/2025).
- [13] Ofer Biran et al. “Heterogeneous Resource Reservation”. In: *2018 IEEE International Conference on Cloud Engineering (IC2E)*. Apr. 2018, pp. 141–147. DOI: 10.1109/IC2E.2018.00037. URL: <https://ieeexplore.ieee.org/document/8360322/?arnumber=8360322> (visited on 12/10/2024).

- [14] Tanaya Biswas and Prashant Kumar. “Optimizing Resource Management in Serverless Computing: A Dynamic Adaptive Scaling Approach”. In: *2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT)*. June 2024, pp. 1–7. DOI: [10.1109/ICCCNT61001.2024.10724128](https://doi.org/10.1109/ICCCNT61001.2024.10724128). URL: <https://ieeexplore.ieee.org/document/10724128/?arnumber=10724128> (visited on 12/10/2024).
- [15] Lukas Böhme et al. “A Penny a Function: Towards Cost Transparent Cloud Programming”. In: *Proceedings of the 2nd ACM SIGPLAN International Workshop on Programming Abstractions and Interactive Notations, Tools, and Environments*. Paint 2023. Cascais, Portugal and New York, NY, USA: Association for Computing Machinery, 2023, pp. 1–10. ISBN: 979-8-4007-0399-7. DOI: [10.1145/3623504.3623566](https://doi.org/10.1145/3623504.3623566). URL: <https://doi.org/10.1145/3623504.3623566>.
- [16] Jon Bryant. “Driving into the Cloud: What Is Finops?” In: *ITNOW* 64.3 (Aug. 2022), pp. 54–55. ISSN: 1746-5702. DOI: [10.1093/combul/bwac097](https://doi.org/10.1093/combul/bwac097). eprint: <https://academic.oup.com/itnow/article-pdf/64/3/54/45484399/bwac097.pdf>. URL: <https://doi.org/10.1093/combul/bwac097>.
- [17] Spot by NetApp. *Cloud Operations Solutions for Optimization & Cost Management*. URL: <https://spot.io/> (visited on 12/18/2024).
- [18] Mohan Baruwat Chhetri et al. “Towards Proactive Risk-Aware Cloud Cost Optimization Leveraging Transient Resources”. In: *IEEE Transactions on Services Computing* 16.4 (July 2023), pp. 3014–3026. ISSN: 1939-1374. DOI: [10.1109/TSC.2023.3253473](https://doi.org/10.1109/TSC.2023.3253473). URL: <https://ieeexplore.ieee.org/document/10061569/?arnumber=10061569> (visited on 12/10/2024).
- [19] Peter Chung. *AWS FinOps Simplified: Eliminate Cloud Waste through Practical FinOps*. Packt Publishing, 2022. ISBN: 978-1-80324-278-1. URL: <https://ieeexplore.ieee.org/document/10162631>.
- [20] *Cloud Billing - Google Cloud*. URL: <https://cloud.google.com/billing/docs/concepts> (visited on 12/18/2024).
- [21] *Cloud Compute Instances – Amazon EC2 Instance Types – AWS*. Amazon Web Services, Inc. URL: <https://aws.amazon.com/ec2/instance-types/> (visited on 06/28/2025).
- [22] *Cloud Cost Analysis - AWS Cost Explorer - AWS*. URL: <https://aws.amazon.com/aws-cost-management/aws-cost-explorer/> (visited on 12/18/2024).
- [23] *Cloud Cost Savings - Savings Plans - AWS*. Amazon Web Services, Inc. URL: <https://aws.amazon.com/savingsplans/> (visited on 06/28/2025).
- [24] *Cloud Optimization Essentials*. URL: <https://www.nops.io/cloud-optimization-essentials/> (visited on 12/18/2024).
- [25] *CloudCostOptimizer*. URL: <https://github.com/AdiY10/CloudCostOptimizer> (visited on 12/19/2024).
- [26] CloudKeeper. *An Essential Guide to Accurate Cloud Cost Forecasting*. 2024. URL: <https://www.cloudkeeper.com/insights/blog/essential-guide-cloud-cost-forecasting>.
- [27] *CloudZero: The Cloud Cost Intelligence Platform*. URL: <https://www.cloudzero.com/> (visited on 12/18/2024).

- [28] *Container Registry - Amazon Elastic Container Registry (Amazon ECR) - AWS*. URL: <https://aws.amazon.com/ecr/> (visited on 06/19/2025).
- [29] *Cost Optimization - AWS Support*. URL: <https://docs.aws.amazon.com/awssupport/latest/user/cost-optimization-checks.html> (visited on 06/23/2025).
- [30] Densify. *A Cloud FinOps Guide to Measuring Cloud Resources for Optimization*. 2024. URL: <https://www.densify.com/finops/cloud>.
- [31] José Luis Díaz et al. “Analysis of the Influence of Per-Second Billing on Virtual Machine Allocation Costs in Public Clouds”. In: *IEEE Transactions on Services Computing* 14.6 (Nov. 2021), pp. 1715–1726. ISSN: 1939-1374. DOI: 10.1109/TSC.2019.2909896. URL: <https://ieeexplore.ieee.org/document/8684322/?arnumber=8684322> (visited on 12/10/2024).
- [32] Quan Ding et al. “A Learning-Based Cost Management System for Cloud Computing”. In: *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)*. Jan. 2018, pp. 362–367. DOI: 10.1109/CCWC.2018.8301738. URL: <https://ieeexplore.ieee.org/document/8301738/?arnumber=8301738> (visited on 12/10/2024).
- [33] Joaquín Entrialgo et al. “Modelling and Simulation for Cost Optimization and Performance Analysis of Transactional Applications in Hybrid Clouds”. In: *Simulation Modelling Practice and Theory* 109 (2021), p. 102311. ISSN: 1569-190X. DOI: 10.1016/j.simpat.2021.102311. URL: <https://www.sciencedirect.com/science/article/pii/S1569190X21000332>.
- [34] Brad Everman, Maxim Gao, and Ziliang Zong. “Evaluating and Reducing Cloud Waste and Cost—A Data-Driven Case Study from Azure Workloads”. In: *Sustainable Computing: Informatics and Systems* 35 (2022), p. 100708. ISSN: 2210-5379. DOI: 10.1016/j.suscom.2022.100708. URL: <https://www.sciencedirect.com/science/article/pii/S2210537922000476>.
- [35] *Factory Method*. URL: <https://refactoring.guru/design-patterns/factory-method> (visited on 06/15/2025).
- [36] *FastAPI*. URL: <https://fastapi.tiangolo.com/> (visited on 06/28/2025).
- [37] *FinOps Cloud Cost Management & Optimization Tool | Finout*. URL: <https://www.finout.io> (visited on 12/18/2024).
- [38] FinOps Foundation. *Cloud Cost Forecasting Playbook*. 2024. URL: <https://www.finops.org/wg/cloud-cost-forecasting/>.
- [39] FinOps Foundation. *What Is FinOps?*
- [40] *Glossary:Enterprise Size*. URL: https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Glossary:Enterprise_size (visited on 06/30/2025).
- [41] *Grafana: The Open and Composable Observability Platform*. URL: <https://grafana.com/> (visited on 06/05/2025).
- [42] *Gwealm/Finops-K8s-Framework*. URL: <https://github.com/gwealm/finops-k8s-framework> (visited on 06/18/2025).
- [43] *How Many Work Days in a Year? (2025)*. URL: <https://www.espocrm.com/blog/how-many-work-days-in-a-year/> (visited on 06/17/2025).

- [44] *How to Find and Resize Azure VM with Low CPU Usage*. URL: <https://www.blinkops.com/blog/finding-and-resizing-azure-virtual-machines-with-low-cpu-usage> (visited on 06/21/2025).
- [45] *Interactive SQL - Amazon Athena - AWS*. URL: <https://aws.amazon.com/athena/> (visited on 06/18/2025).
- [46] Pete Rathburn Full Bio Pete Rathburn is a copy editor et al. *Empirical Rule: Definition, Formula, and Example*. URL: <https://www.investopedia.com/terms/e/empirical-rule.asp> (visited on 06/27/2025).
- [47] Mike Fuller J.R. Storment. *Cloud FinOps*. 2nd ed. O'Reilly Media, Inc., 2023. ISBN: 978-1-4920-9835-5.
- [48] Harshavardhan Kadiyala, Alberto Misail, and Julia Rubin. "Kuber: Cost-Efficient Microservice Deployment Planner". In: *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. Mar. 2022, pp. 252–262. DOI: 10.1109/SANER53432.2022.00040. URL: <https://ieeexplore.ieee.org/document/9825903/?arnumber=9825903> (visited on 12/10/2024).
- [49] Murat Karslioglu. "Kubernetes - a Complete DevOps Cookbook". In: Packt Publishing Ltd, 2020.
- [50] Akif Quddus Khan et al. "Towards Graph-based Cloud Cost Modelling and Optimisation". In: *2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)*. June 2023, pp. 1337–1342. DOI: 10.1109/COMPSAC57700.2023.00203. URL: <https://ieeexplore.ieee.org/document/10197049/?arnumber=10197049> (visited on 12/10/2024).
- [51] *Kind*. URL: <https://kind.sigs.k8s.io/> (visited on 06/17/2025).
- [52] *Kubernetes Components*. Kubernetes. URL: <https://kubernetes.io/docs/concepts/overview/components/> (visited on 06/28/2025).
- [53] *Laws of Large Number - an Overview | ScienceDirect Topics*. URL: <https://www.sciencedirect.com/topics/computer-science/laws-of-large-number> (visited on 06/27/2025).
- [54] Colin David Lewis. *Industrial and Business Forecasting Methods*. London: Butterworth-Heinemann, 1982. ISBN: 978-0-408-00559-3.
- [55] Fang Li et al. "SmartCMP: A Cloud Cost Optimization Governance Practice of Smart Cloud Management Platform". In: *2022 IEEE 7th International Conference on Smart Cloud (SmartCloud)*. Oct. 2022, pp. 171–176. DOI: 10.1109/SmartCloud55982.2022.00034. URL: <https://ieeexplore.ieee.org/document/9944800/?arnumber=9944800> (visited on 12/10/2024).
- [56] *LocalStack*. URL: <https://www.localstack.cloud/> (visited on 06/07/2025).
- [57] *Managed Kubernetes - Amazon Elastic Kubernetes Service (EKS) - AWS*. Amazon Web Services, Inc. URL: <https://aws.amazon.com/eks/> (visited on 06/28/2025).
- [58] Artan Mazrekaj, Isak Shabani, and Besmir Sejdiu. "Pricing Schemes in Cloud Computing: An Overview". In: *International Journal of Advanced Computer Science and Applications* 7 (Feb. 2016). DOI: 10.14569/IJACSA.2016.070211.
- [59] Peter M. Mell and Timothy Grance. *SP 800-145. The NIST Definition of Cloud Computing*. Gaithersburg, MD, USA: National Institute of Standards & Technology / National Institute of Standards and Technology, 2011.

- [60] *Microsoft Cost Management - Microsoft Azure*. URL: <https://azure.microsoft.com/en-us/products/cost-management> (visited on 12/18/2024).
- [61] Steven Gonzalez Monserrate. “The Cloud Is Material: On the Environmental Impacts of Computation and Data Storage”. In: *MIT Case Studies in Social and Ethical Responsibilities of Computing* (Winter 2022 Jan. 27, 2022).
- [62] Ganesh Kumar Murugesan. “Cloud Cost Factors and AWS Cost Optimization Techniques”. In: *2024 12th International Symposium on Digital Forensics and Security (ISDFS)*. Apr. 2024, pp. 1–7. DOI: 10.1109/ISDFS60797.2024.10527314. URL: <https://ieeexplore.ieee.org/document/10527314/?arnumber=10527314> (visited on 12/10/2024).
- [63] Mageshkumar Naarayanasamy Varadarajan et al. “Ai-Powered Financial Operation Strategy for Cloud Computing Cost Optimization for Future;” in: *Salud, Ciencia y Tecnologia - Serie de Conferencias 3* (2024). ISSN: 29534860. DOI: 10.56294/sctconf2024694. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85198634679&doi=10.56294%2fsctconf2024694&partnerID=40&md5=ee747f215d9610968ef608084737e12c>.
- [64] *OpenCost — Open Source Cost Monitoring for Cloud Native Environments* | *OpenCost — Open Source Cost Monitoring for Cloud Native Environments*. URL: <https://opencost.io/> (visited on 06/05/2025).
- [65] Matthew J. Page et al. “The PRISMA 2020 Statement: An Updated Guideline for Reporting Systematic Reviews”. In: *BMJ (Clinical research ed.)* 372 (Mar. 2021), n71. ISSN: 1756-1833. DOI: 10.1136/bmj.n71. PMID: 33782057. URL: <https://www.bmj.com/content/372/bmj.n71> (visited on 12/15/2024).
- [66] Sivakumar Ponnusamy and Mandar Khoje. “Optimizing Cloud Costs with Machine Learning: Predictive Resource Scaling Strategies”. In: *2024 5th International Conference on Innovative Trends in Information Technology (ICITIIT)*. Mar. 2024, pp. 1–8. DOI: 10.1109/ICITIIT61487.2024.10580717. URL: <https://ieeexplore.ieee.org/document/10580717/?arnumber=10580717> (visited on 12/10/2024).
- [67] *Prometheus - Monitoring System & Time Series Database*. URL: <https://prometheus.io/> (visited on 06/05/2025).
- [68] *Reserved Instances - Amazon EC2 Reserved Instances - AWS*. Amazon Web Services, Inc. URL: <https://aws.amazon.com/ec2/pricing/reserved-instances/> (visited on 06/28/2025).
- [69] *Resource Quotas*. Kubernetes. URL: <https://kubernetes.io/docs/concepts/policy/resource-quotas/> (visited on 06/29/2025).
- [70] Seol Roh et al. “An Efficient Serverless-VM Switching Mechanism for Cloud Cost Optimization”. In: *Proceedings of the 2024 9th International Conference on Intelligent Information Technology*. Ho Chi Minh City Vietnam: ACM, Feb. 2024, pp. 482–486. ISBN: 979-8-4007-1671-3. DOI: 10.1145/3654522.3654594. URL: <https://dl.acm.org/doi/10.1145/3654522.3654594> (visited on 12/10/2024).
- [71] *Serverless Compute - AWS Fargate - AWS*. Amazon Web Services, Inc. URL: <https://aws.amazon.com/fargate/> (visited on 06/28/2025).
- [72] Amazon Web Services. *Analyzing Your Costs and Usage with AWS Cost Explorer*. URL: <https://docs.aws.amazon.com/cost-management/latest/userguide/ce-what-is.html>.

- [73] Simeen Sheikh, G. Suganya, and M. Premalatha. “Automated Resource Management on AWS Cloud Platform”. In: *Smart Innovation, Systems and Technologies* 164 (2020). Ed. by Vijayakumar V et al., pp. 133–147. ISSN: 21903018. DOI: 10.1007/978-981-32-9889-7_11. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85075574052&doi=10.1007%2f978-981-32-9889-7_11&partnerID=40&md5=a5929d177978291371a2be78f00da150.
- [74] Mukesh Sevak Shende and Manoj B. Chandak. “Cost Optimization of Cloud Services through Automated Analytics and Resource Allocation”. In: *15th International Conference on Advances in Computing, Control, and Telecommunication Technologies, ACT 2024*. Ed. by Stephen J et al. Vol. 2. Grenze Scientific Society, 2024, pp. 1727–1732. ISBN: 979-8-3313-0057-9. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85209151957&partnerID=40&md5=96c7537f06fe8da2f96c8d00d0c55de3>.
- [75] *Shift FinOps Left with Infracost*. URL: <https://www.infracost.io/> (visited on 12/18/2024).
- [76] *SIGNEXT/Aws-Cost-Estimator*. GitHub. URL: <https://github.com/SIGNEXT/aws-cost-estimator> (visited on 07/08/2025).
- [77] *Singleton*. URL: <https://refactoring.guru/design-patterns/singleton> (visited on 06/20/2025).
- [78] *Singleton in Python / Design Patterns*. URL: <https://refactoring.guru/design-patterns/singleton/python/example> (visited on 06/24/2025).
- [79] Mateusz Smendowski and Piotr Nawrocki. “Optimizing Multi-Time Series Forecasting for Enhanced Cloud Resource Utilization Based on Machine Learning”. In: *Knowledge-Based Systems* 304 (2024), p. 112489. ISSN: 0950-7051. DOI: 10.1016/j.knosys.2024.112489. URL: <https://www.sciencedirect.com/science/article/pii/S0950705124011237>.
- [80] Xurui Song, Li Pan, and Shijun Liu. “An Online Algorithm for Optimally Releasing Multiple On-Demand Instances in IaaS Clouds”. In: *Future Generation Computer Systems* 136 (2022), pp. 311–321. ISSN: 0167-739X. DOI: 10.1016/j.future.2022.06.014. URL: <https://www.sciencedirect.com/science/article/pii/S0167739X22002242>.
- [81] Barrie Sosinsky. “Cloud Computing Bible”. In: 1st ed. Wiley Publishing, 2011, pp. 3–23. ISBN: 978-0-470-90356-8.
- [82] *Spot Instance Interruptions - Amazon Elastic Compute Cloud*. URL: <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/spot-interruptions.html> (visited on 06/28/2025).
- [83] Animesh Srivastava et al. “Innovative Cloud Service Monitoring Techniques for Optimal Performance”. In: *2024 IEEE 13th International Conference on Communication Systems and Network Technologies (CSNT)*. Apr. 2024, pp. 514–518. DOI: 10.1109/CSNT60213.2024.10545799. URL: <https://ieeexplore.ieee.org/document/10545799/?arnumber=10545799> (visited on 12/10/2024).
- [84] *Stackrox/Stackrox*. June 2025. URL: <https://github.com/stackrox/stackrox> (visited on 06/17/2025).
- [85] *Storage | Prometheus*. URL: <https://prometheus.io/docs/prometheus/latest/storage/> (visited on 06/29/2025).

- [86] *Sustainable Cloud Computing - Amazon Web Services*. Amazon Web Services, Inc. URL: <https://aws.amazon.com/sustainability/> (visited on 12/18/2024).
- [87] *The FinOps Foundation - The State of FinOps*. URL: <https://data.finops.org> (visited on 12/18/2024).
- [88] *Understand the Kubernetes Version Lifecycle on EKS - Amazon EKS*. URL: <https://docs.aws.amazon.com/eks/latest/userguide/kubernetes-versions.html> (visited on 06/28/2025).
- [89] *Understanding Rightsizing Recommendations Calculations - AWS Cost Management*. URL: <https://docs.aws.amazon.com/cost-management/latest/userguide/understanding-rr-calc.html> (visited on 06/21/2025).
- [90] Xi Yang et al. "Optimizing IT FinOps and Sustainability through Unsupervised Workload Characterization". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Ed. by Wooldridge M, Dy J, and Natarajan S. Vol. 38. Association for the Advancement of Artificial Intelligence, 2024, pp. 22990–22996. DOI: 10.1609/aaai.v38i21.30340. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85189639312&doi=10.1609%2faaaai.v38i21.30340&partnerID=40&md5=83ac64a434b8f85a3f640e6a11aa8f00>.
- [91] Adi Yehoshua, Ilya Kolchinsky, and Assaf Schuster. "CCO - Cloud Cost Optimizer". In: *Proceedings of the 16th ACM International Conference on Systems and Storage, SYSTOR 2023*. 1601 Broadway, 10th Floor, NEW YORK, NY, UNITED STATES: Association for Computing Machinery, Inc, 2023, p. 137. ISBN: 978-1-4503-9962-3. DOI: 10.1145/3579370.3594746. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85165989173&doi=10.1145%2f3579370.3594746&partnerID=40&md5=2e757a6ee1786cb5656f477f99c01d2d>.

Appendix A

Selected Studies

Table A.1: Selected Studies (2018-2020)

Reference	Research Purpose	Approach/Domain
Ding et al. (2018) [32]	Design a cost management system that automatically monitors the cloud environment, tracks changes in relevant metrics near real-time and changes.	Real-Time Monitoring, Machine Learning, Cost Management
Biran et al. (2018) [13]	Introduce a framework to optimize heterogeneous resource allocation and billing contracts for elastic workloads.	Billing Optimization, Resource Management, Cost Minimization
Sheikh et al. (2020) [73]	Automates resource management on AWS using tools like AWS Lambda and Infrastructure as Code (IaC).	Automation, AWS, Resource Management

Table A.2: Selected Studies (2021)

Reference	Research Purpose	Approach/Domain
Entrialgo et al. (2021) [33]	Optimize resource distribution in hybrid cloud environments using simulation-based performance analysis and cost optimization strategies.	Hybrid Clouds, VM Allocation
Díaz et al. (2021) [31]	Analyze the cost implications of per-second billing on virtual machine allocation strategies.	Billing Models, Cost Analysis

Table A.3: Selected Studies (2022)

Reference	Research Purpose	Approach/Domain
Chung (2022) [19]	Implement FinOps practices in AWS to identify and eliminate cloud waste.	FinOps, AWS
Song et al. (2022) [80]	Propose an online algorithm for cost-efficient release of on-demand cloud instances, optimizing resource utilization for SaaS providers	Online Algorithms, IaaS Cost Management
Everman et al. (2022) [34]	Evaluate and reduce cloud waste through a data-driven analysis of Azure workloads.	Cloud Waste, Data Analytics
Kadiyala et al. (2022) [48]	Develop a microservice deployment planner (KUBER) that optimizes resource allocation for cost efficiency in cloud environments.	Microservices, VM Optimization
Li et al. (2022) [55]	Propose a framework (SmartCMP) for real-time cloud cost analysis and optimization.	Real-Time Monitoring, Cloud Cost Optimization

Table A.4: Selected Studies (2023)

Reference	Research Purpose	Approach/Domain
Yehoshua et al. (2023) [91]	Showcase a scalable Cloud Cost Optimizer (CCO) to improve efficiency in hybrid and public cloud environments.	Cost Optimization, Hybrid-Cloud, Meta-Heuristics
Chhetri et al. (2023) [18]	Introduce risk-aware cost optimization techniques leveraging transient cloud resources.	Risk Management, Transient Resources, Portfolio Diversification
Khan et al. (2023) [50]	Propose graph-based cost modeling and optimization techniques for cloud environments.	Cost Modeling, Cost Optimization

Table A.5: Selected Studies (2024)

Reference	Research Purpose	Approach/Domain
Smendowski and Nawrocki (2024) [79]	Enhance cloud resource utilization by employing multi-time series forecasting techniques to optimize resource allocation and achieve significant cost savings in dynamic cloud environments.	Machine Learning, Time Series Forecasting, Cost Optimization
Roh et al. (2024) [70]	Design a serverless-VM switching mechanism to enhance cost management and resource utilization.	Switching Mechanism, Cost Optimization, Resource Management
Ponnusamy and Khoje (2024) [66]	Develop predictive resource scaling strategies using machine learning to improve cloud cost optimization.	Cost Optimization, Predictive Analytics, Machine Learning
Shende and Chandak (2024) [74]	Implement a framework that leverages predictive analytics, dynamic resource allocation strategies and IaC mechanisms to dynamically optimize cloud costs.	Predictive Analytics, IaC, Resource Optimization
Yang et al. (2024) [90]	Develop an unsupervised machine learning framework for automatically identifying resources that can be shutdown or scheduled more efficiently.	Workload Characterization, Unsupervised Learning, Resource Optimization
Biswas and Kumar (2024) [14]	Develop a dynamic adaptive scaling model for serverless computing to enhance resource management and reduce costs through realtime monitoring and predictive analytics.	Serverless Computing, Dynamic Resource Management, Predictive Analytics, Machine Learning
Murugesan (2024) [62]	Provide an overview of AWS cost optimization techniques.	AWS, Cost Optimization
Srivastava et al. (2024) [83]	Develop innovative monitoring methods to optimize cloud service performance and cost efficiency.	Monitoring, Performance Optimization
Naarayanasamy Varadarajan et al. (2024) [63]	Leverage AI and machine learning to automate financial strategies for cloud cost management.	AI, Automation, Cost Optimization

Appendix B

Implementation Details

This chapter provides implementation details, such as pseudocode snippets, configuration files, and scripts used in the FinOps frameworks presented during the course of this work.

B.1 Kubernetes-based Framework

B.1.1 Pseudocode Snippets

Algorithm 5 Statistical Analysis and Anomaly Score Calculation

```

1: procedure CALCULATEANOMALYSCORE(hist_data, current_cost, namespace)
2:   data_points  $\leftarrow$  EXTRACTHISTORICALVALUES(hist_data)
3:   if |data_points| > 24 then                                 $\triangleright$  Ensure there are at least 24 data points
4:     historical_mean  $\leftarrow$  CALCULATEMEAN(data_points)
5:     cost_std  $\leftarrow$  CALCULATESTANDARDDEVIATION(data_points)
6:     if cost_std > 0 then
7:       z_score  $\leftarrow$   $\left| \frac{\text{current\_cost} - \text{historical\_mean}}{\text{cost\_std}} \right|$ 
8:       anomaly_score  $\leftarrow$  min(100, max(0, (z_score - 1)  $\times$  50))
9:     else
10:      if current_cost = historical_mean then
11:        anomaly_score  $\leftarrow$  0
12:      else
13:        anomaly_score  $\leftarrow$  100
14:      end if
15:    end if
16:    if historical_mean > 0 then
17:      increase_pct  $\leftarrow$   $\frac{\text{current\_cost} - \text{historical\_mean}}{\text{historical\_mean}} \times 100$ 
18:    else
19:      increase_pct  $\leftarrow$  0
20:    end if
21:  else
22:    anomaly_score  $\leftarrow$  0
23:    increase_pct  $\leftarrow$  0
24:    historical_mean  $\leftarrow$  current_cost
25:  end if
26:  UPDATEANOMALYMETRIC(namespace, anomaly_score)
27:  return anomaly_score, increase_pct, historical_mean
28: end procedure

```

Algorithm 6 Efficiency Score Calculation

```

1: procedure CALCULATEEFFICIENCYSCORE(data)
2:   if data.cpu_request > 0 then
3:     cpu_waste  $\leftarrow \max\left(0, \frac{\text{data.cpu\_request} - \text{data.cpu\_usage}}{\text{data.cpu\_request}}\right) \times 100$ 
4:   else
5:     cpu_waste  $\leftarrow 0$  ▷ No waste if no resources requested
6:   end if
7:   if data.memory_request > 0 then
8:     mem_waste  $\leftarrow \max\left(0, \frac{\text{data.memory\_request} - \text{data.memory\_usage}}{\text{data.memory\_request}}\right) \times 100$ 
9:   else
10:    mem_waste  $\leftarrow 0$  ▷ No waste if no resources requested
11:  end if
12:  score  $\leftarrow 100 - \frac{\text{cpu\_waste} + \text{mem\_waste}}{2}$ 
13:  UPDATEMETRICS(data.namespace, score, cpu_waste, mem_waste)
14:  return COSTEFFICIENCY(data.namespace, score, cpu_waste, mem_waste)
15: end procedure

```

B.1.2 Kind Deployment Script

Listing B.1: Kind Deployment Script

```

1 echo "Creating Kind cluster..."
2 kind create cluster --name finops-poc --config kubernetes/kind-config.yaml
3
4 echo "Pre-loading images..."
5 docker pull ghcr.io/opencost/opencost:1.114.0
6 docker pull ghcr.io/opencost/opencost-ui:1.114.0
7 kind load docker-image ghcr.io/opencost/opencost:1.114.0 --name finops-poc
8 kind load docker-image ghcr.io/opencost/opencost-ui:1.114.0 --name finops-poc
9
10 # Create namespaces
11 kubectl create namespace finops
12 kubectl create namespace monitoring
13 kubectl create namespace opencost
14
15 echo "Installing NGINX Ingress Controller..."
16 kubectl apply -f https://kind.sigs.k8s.io/examples/ingress/deploy-ingress-nginx.
    yaml
17
18 echo "Waiting for ingress controller to be ready..."
19 kubectl wait --namespace ingress-nginx \
20   --for=condition=ready pod \
21   --selector=app.kubernetes.io/component=controller \
22   --timeout=90s

```

```
23
24 # Create a configmap for the custom Grafana dashboards
25 kubectl create configmap custom-grafana-dashboards \
26   --from-file=grafana/dashboards/ \
27   -n monitoring
28
29 # Label the configmap with the expected label
30 kubectl label configmap custom-grafana-dashboards grafana_dashboard=1 -n monitoring
31
32 # Install OpenCost and Prometheus Stack
33 echo "Adding Helm repositories..."
34 helm repo add prometheus-community https://prometheus-community.github.io/helm-
    charts
35 helm repo add opencost https://opencost.github.io/opencost-helm-chart
36 helm repo update
37
38 echo "Installing Prometheus Stack (Prometheus, Grafana, AlertManager)..."
39 helm install prometheus prometheus-community/kube-prometheus-stack \
40   --namespace monitoring --create-namespace \
41   -f values.yaml
42
43 echo "Waiting for Prometheus components to start..."
44 kubectl wait --for=condition=available --timeout=300s deployment/prometheus-kube-
    prometheus-operator -n monitoring
45
46 # Installing OpenCost
47 helm install opencost opencost/opencost \
48   --namespace opencost --create-namespace \
49   --set opencost.prometheus.internal.namespaceName="monitoring" \
50   --set opencost.prometheus.internal.port=9090 \
51   --set opencost.prometheus.internal.serviceName="prometheus-kube-prometheus-
    prometheus" \
52   --set opencost.metrics.serviceMonitor.enabled="true" \
53   --set opencost.metrics.serviceMonitor.additionalLabels.release=prometheus
54
55 echo "Waiting for OpenCost to start..."
56 kubectl wait --for=condition=available --timeout=300s deployment/opencost -n
    opencost
57
58 echo "Applying FinOps alert rules..."
59 kubectl apply -f finops-alerts.yml
60
61 echo "Waiting for ingress controller to be ready..."
62 kubectl wait --namespace ingress-nginx \
63   --for=condition=ready pod \
64   --selector=app.kubernetes.io/component=controller \
65   --timeout=90s
66
67 echo "Applying Ingress configuration..."
```

```

68 kubectl apply -f kubernetes/opencost.yaml
69 kubectl apply -f kubernetes/monitoring.yaml
70
71 echo "Deploying test PVC..."
72 kubectl apply -f pvc-test.yaml
73
74 echo "Checking test PVC status..."
75 kubectl wait --for=condition=Ready pod/pod-using-pvc -n monitoring --timeout=60s
76 if [ $? -eq 0 ]; then
77     echo "PVC test successful, pod is running"
78 else
79     echo "Warning: PVC test pod is not ready, may need troubleshooting"
80 fi
81
82 echo "Building FinOps API Docker Image..."
83 docker build -t finops-api:latest ./app/
84
85 echo "Loading FinOps API Docker Image into Kind..."
86 kind load docker-image finops-api:latest --name finops-poc
87
88 echo "Deploying FinOps API to Kubernetes..."
89 kubectl apply -f kubernetes/finops-api.yaml
90
91 echo "Waiting for FinOps API to be ready for use..."
92 kubectl wait --for=condition=available --timeout=300s deployment/finops-api -n
    finops
93
94 echo "Adding hosts to /etc/hosts..."
95 NODE_IP=$(kubectl get nodes -o wide | awk 'NR==2{print $6}')
96 HOSTS_ENTRY="$NODE_IP grafana prometheus alertmanager opencost opencost-api finops-
    api"
97
98 # Check if the entry already exists
99 if ! grep -q "grafana prometheus alertmanager opencost opencost-api finops-api" /
    etc/hosts; then
100     echo "$HOSTS_ENTRY" | sudo tee -a /etc/hosts
101 else
102     echo "Hosts entry already exists, skipping..."
103 fi
104
105 # Get the Grafana password
106 GRAFANA_PASSWORD=$(kubectl --namespace monitoring get secrets prometheus-grafana -o
    jsonpath='{.data.admin-password}' | base64 -d ; echo)
107
108 echo "Setup complete!"
109 echo ""
110 echo "To access Grafana:"
111 echo "http://grafana"
112 echo "Username: admin"

```

```
113 echo "Password: $GRAFANA_PASSWORD"
114 echo ""
115 echo "To access OpenCost UI:"
116 echo "http://opencost"
117 echo ""
118 echo "To access OpenCost API:"
119 echo "http://opencost-api"
120 echo ""
121 echo "To access Prometheus:"
122 echo "http://prometheus"
123 echo ""
124 echo "To access Alertmanager:"
125 echo "http://alertmanager"
126 echo ""
127 echo "FinOps Enhanced API deployment complete!"
128 echo ""
129 echo "Access the API at: http://finops-api:8000"
130 echo "API Documentation: http://finops-api:8000/docs"
131 echo ""
132 echo "Try these endpoints:"
133 echo "- /cost-efficiency          - Get cost efficiency scores by namespace"
134 echo "- /recommendations          - Get cost optimization recommendations"
135 echo "- /cost-anomalies           - Get cost anomaly detection results"
136 echo "- /cost-forecasts           - Get cost forecasts by namespace"
137 echo "- /all-insights              - Get all insights in one call"
138 echo "- /update-metrics            - Force an update of the Prometheus metrics"
139 echo ""
140 echo "Custom metrics are available in Prometheus under these names:"
141 echo "- finops_efficiency_score - Cost efficiency score by namespace (0-100%)"
142 echo "- finops_resource_waste - Percentage of wasted resources by type and
    namespace"
143 echo "- finops_anomaly_score - Cost anomaly detection score by namespace"
144 echo "- finops_optimization_savings - Potential monthly cost savings by
    recommendation type"
145 echo "- finops_cost_forecast - 30-day cost forecast by namespace"
146 echo "- finops_resource_utilization - Resource utilization ratio by type and
    namespace"
```

B.2 CloudWatch-based Framework

B.2.1 Pseudocode Snippets

Algorithm 7 Main Efficiency Score Calculation

```

1: procedure CALCULATEEFFICIENCYSCORE(metrics, weights)
2:                                     ▷ Step 1: Calculate and normalize metrics
3:   cpu_util  $\leftarrow$  mean(metrics.CPUUtilization)
4:   memory_util  $\leftarrow$  mean(metrics.MemoryUtilization)
5:   network_in  $\leftarrow$  mean(metrics.NetworkIn)
6:   network_out  $\leftarrow$  mean(metrics.NetworkOut)
7:   normalized_memory  $\leftarrow \frac{\text{memory\_util}}{100}$ 
8:   normalized_cpu  $\leftarrow \frac{\text{cpu\_util}}{100}$ 
9:                                     ▷ Step 2: Network normalization using observed maximum
10:  total_network  $\leftarrow$  network_in + network_out
11:  max_observed  $\leftarrow$  GETMAXOBSERVEDNETWORK(instance_type)
12:  if max_observed > 0 then
13:    normalized_network  $\leftarrow$  min( $\frac{\text{total\_network}}{\text{max\_observed}}$ , 1.0)
14:  else
15:    normalized_network  $\leftarrow$  0
16:  end if
17:                                     ▷ Step 3: Apply configured weights
18:  efficiency  $\leftarrow$  (weights.cpu  $\times$  normalized_cpu +
19:    weights.memory  $\times$  normalized_memory +
20:    weights.network  $\times$  normalized_network)  $\times$  100
21:  return min(100, max(0, efficiency))
22: end procedure

```

Algorithm 8 Find Optimal Instance

```

1: procedure FINDOPTIMALINSTANCE(current_instance, metrics, network_usages)
2:   current_specs  $\leftarrow$  instance_catalog[current_instance.type]
3:   safety_margin  $\leftarrow$  1.3 ▷ 30% buffer from config
4:   ▷ Calculate required resources with safety margin
5:   cpu_util  $\leftarrow$  mean(metrics.CPUUtilization)
6:   mem_util  $\leftarrow$  mean(metrics.MemoryUtilization)
7:   total_network  $\leftarrow$  mean(metrics.NetworkIn + metrics.NetworkOut)
8:   required_vcpus  $\leftarrow$   $\lceil (\frac{cpu\_util}{100}) \times current\_specs.vcpu \times safety\_margin \rceil$ 
9:   required_memory  $\leftarrow$   $\lceil (\frac{mem\_util}{100}) \times current\_specs.memory \times safety\_margin \rceil$ 
10:  suitable_instances  $\leftarrow$  []
11:  for each (type, specs) in instance_catalog do
12:    if type = current_instance.type then
13:      continue ▷ Skip current type
14:    end if
15:    if specs.vcpu  $\geq$  required_vcpus AND specs.memory  $\geq$  required_memory then
16:      ▷ Verify network capacity based on observed usage
17:      if type in network_usages then
18:        max_network  $\leftarrow$  max(network_usages[type])
19:        if max_network < total_network  $\times$  safety_margin then
20:          continue ▷ Insufficient network
21:        end if
22:      end if
23:      suitable_instances.append((type, specs.cost_per_hour))
24:    end if
25:  end for
26:  if suitable_instances is empty then
27:    return null
28:  end if
29:  ▷ Select instance with minimum cost
30:  cheapest  $\leftarrow$  MINBYCOST(suitable_instances)
31:  if cheapest.cost  $\geq$  current_specs.cost_per_hour then
32:    return null ▷ No savings available
33:  end if
34:  return cheapest.type
35: end procedure

```

B.2.2 Configuration Files

The files presented in this section are used to configure the FinOps framework presented in Chapter 5. Note that the values presented in here are just for demonstration purposes and should be adjusted to your specific use case, as explained in Section 5.2.4.

Listing B.2: CloudWatch-based Framework Service-Specific Configuration File

```

1 analysis:
2   thresholds:
3     utilization_ratio:
4       cpu:
5         low: 0.2 # Using 20% of CPU capacity
6         very_low: 0.01 # Using 5% of CPU capacity
7         target_min: 0.4 # Ideally using at least 40% of capacity
8         target_max: 0.8 # Ideally not exceeding 80% of capacity
9       memory:
10        low: 0.3 # Using 30% of memory capacity
11        very_low: 0.1 # Using 10% of memory capacity
12        target_min: 0.5 # Ideally using at least 50% of capacity
13        target_max: 0.8 # Ideally not exceeding 80% of capacity
14      network:
15        low: 0.1 # Using 10% of available bandwidth
16        very_low: 0.01 # Using 1% of available bandwidth
17
18    rightsizing:
19      min_savings: 0.0 # Minimum $ savings per month to recommend changes
20      safety_margin: 1.3 # 30% buffer for rightsizing recommendations
21      efficiency_threshold_very_low: 10.0 # Recommend change if below this
22        efficiency
23      efficiency_threshold_low: 30.0 # Recommend change if below this efficiency
24    weights:
25      cpu: 0.4
26      memory: 0.3
27      network: 0.3
28
29  mocks:
30    # profiles will be used in any mode except aws or if the flag --pure-csv is
31    # passed
32  profiles:
33    profile_example:
34      description: "Example description for the profile type"
35      distribution: 3 # Number of instances with this profile
36      instance_types: ["m5.large", "m5.xlarge", "r5.large"] # the instances will
37        be chosen randomly from this types
38      region: "eu-central-1"
39      metrics:
40        CPUUtilization:

```

```
38         range: [2, 15]
39     MemoryUtilization:
40         range: [5, 25]
41     NetworkIn:
42         range: [10000, 500000]
43     NetworkOut:
44         range: [5000, 200000]
```

Listing B.3: CloudWatch-based Framework Global Configuration File

```
1 global:
2     monitoring:
3         collection_interval: 300 # seconds
4         prometheus_port: 8000
```

Appendix C

Grafana Dashboards

C.1 Kubernetes-based Framework

C.1.1 Dashboards - OpenCost / Overview

This section features screenshots of the Grafana overview dashboard, which provides a comprehensive view of the cluster's cost metrics and resource utilization.

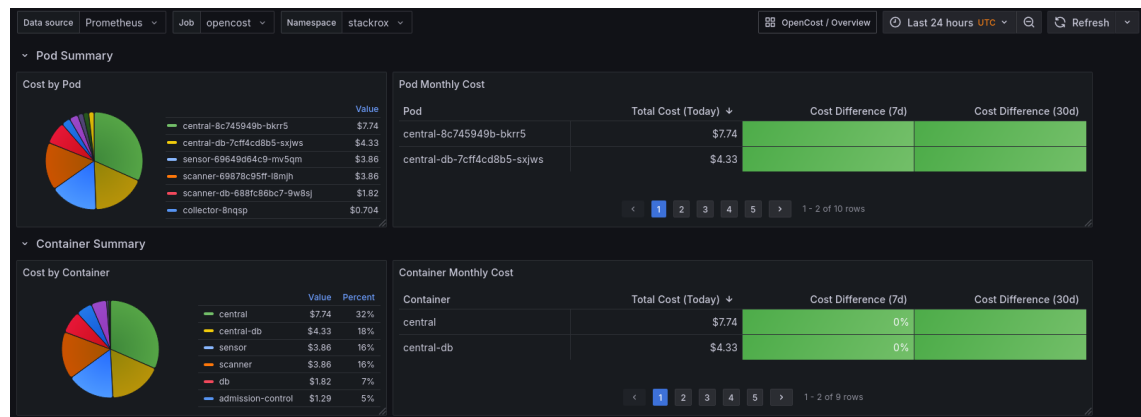


Figure C.1: Pod and Container Summary dashboard showing cost breakdowns by pod and container, including cost distribution visualizations, monthly cost analysis with total costs and percentage-based cost variance indicators, and detailed cost metrics for individual pods and containers. The 30-day cost difference values are not displayed as the framework requires at least 30 days of data to calculate them.

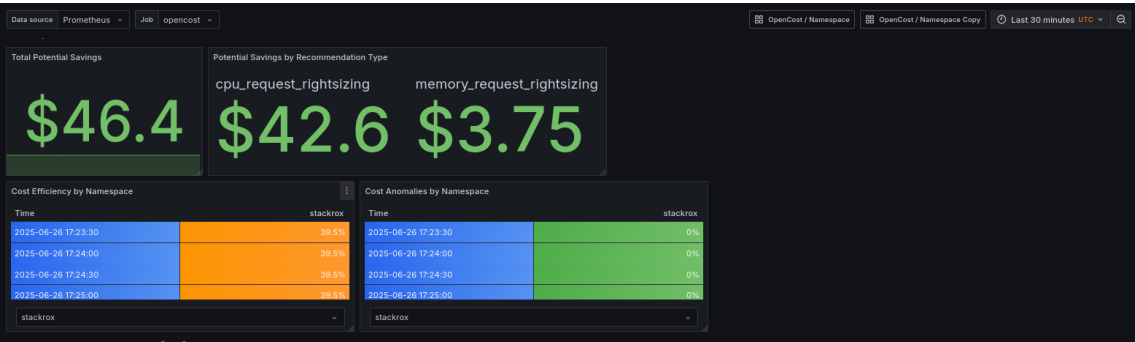


Figure C.2: FinOps Middleware dashboard showing the Total Potential Savings view with savings opportunities across CPU and memory request rightsizing recommendations, cost efficiency score, and cost anomaly score by namespace with temporal analysis over time.

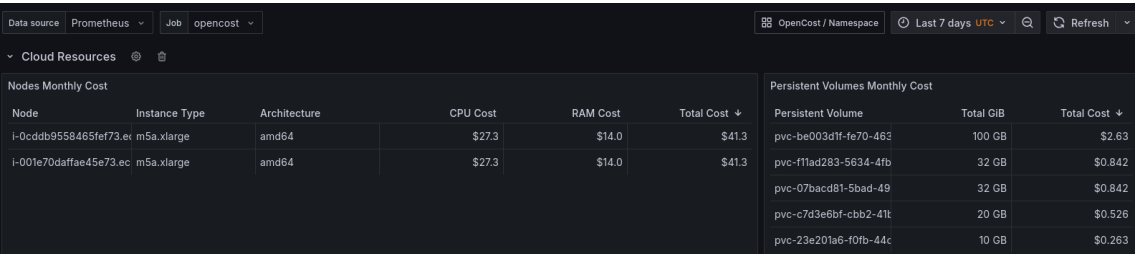


Figure C.3: Cloud Resources dashboard displaying nodes monthly cost breakdown with information about the instance type, architecture, CPU cost, RAM cost, and total cost, alongside persistent volumes monthly cost information showing volume names, storage capacity, and associated costs.

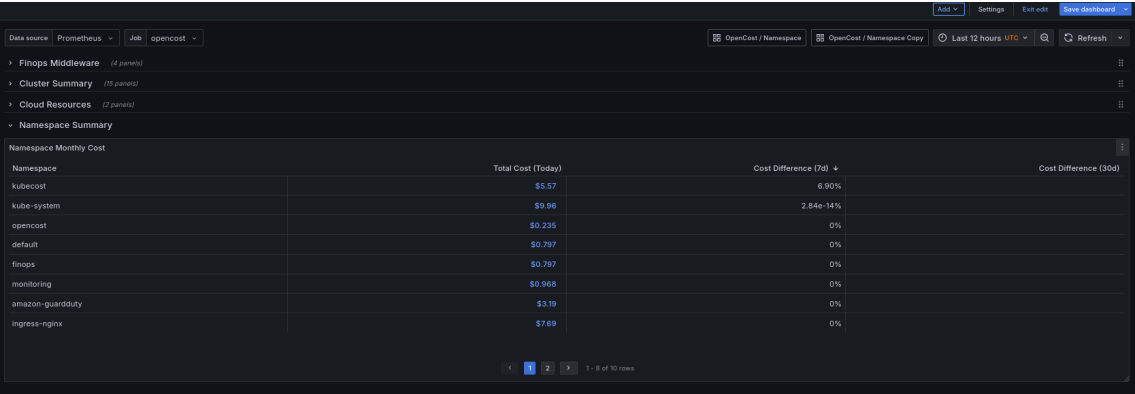


Figure C.4: Namespace Summary dashboard displaying namespace monthly cost breakdown with total costs and percentage-based cost variance indicators across different namespaces.

C.1.2 Dashboards - OpenCost / Namespace

This section features screenshots of the Grafana namespace dashboard, which provides a detailed view of costs and resource utilization for each namespace within the cluster.

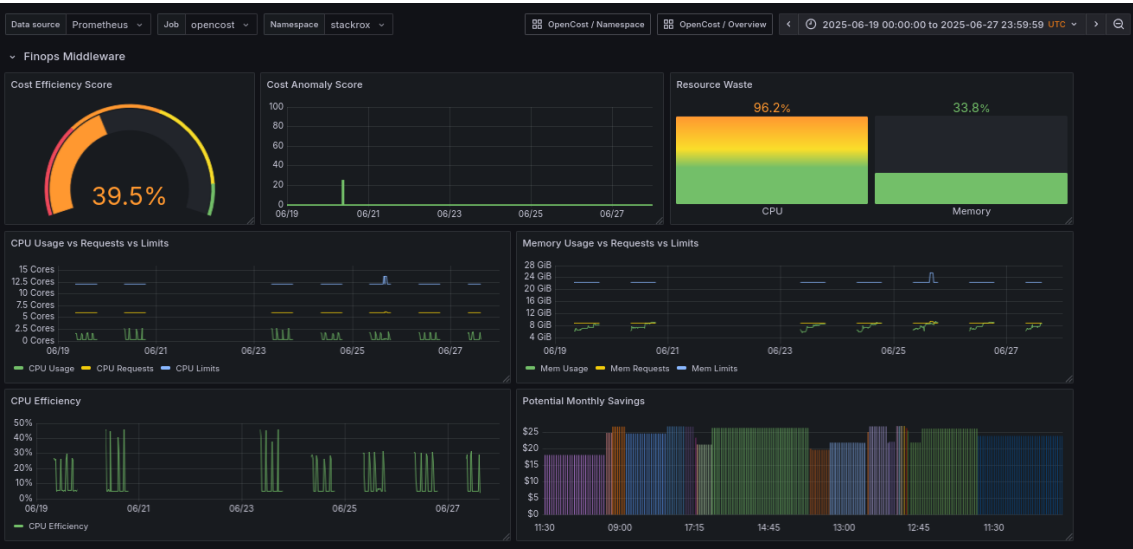


Figure C.5: FinOps Middleware dashboard displaying CPU and Memory usages versus requests and limits, cost efficiency scores, cost anomaly scores, resource waste percentages, CPU efficiency trends, and potential monthly savings.

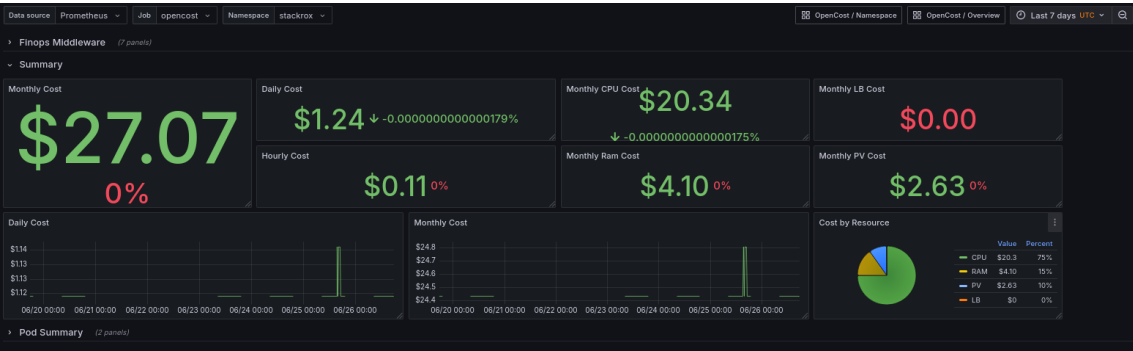


Figure C.6: Summary dashboard presenting key financial metrics including total monthly, daily, and hourly costs, monthly cost breakdowns by resource, and monthly costs of Persistent Volumes and Load Balancers.

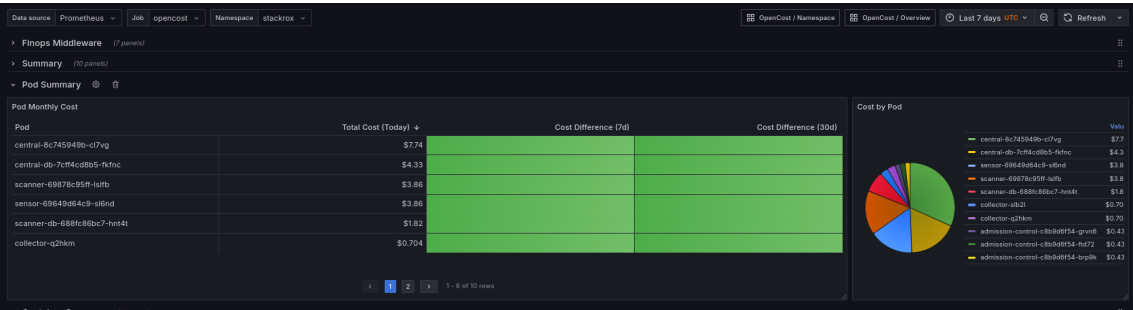


Figure C.7: Pod Summary dashboard presenting monthly cost analysis by individual pods, cost variance indicators across different time periods, and a cost distribution breakdown by pod.



Figure C.8: Container Summary dashboard presenting monthly cost analysis by individual containers, cost variance indicators across different time periods, and a cost distribution breakdown by container.

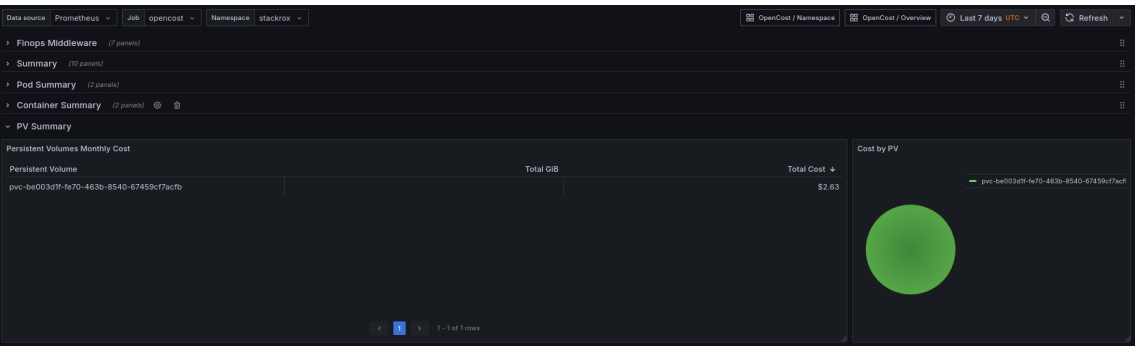


Figure C.9: PV Summary dashboard presenting persistent volumes monthly cost analysis with total storage capacity metrics, individual volume cost breakdowns, and cost distribution visualization across different persistent volume resources. In this scenario, only one persistent volume is configured in the system.

C.1.3 Dashboards - OpenCost / Node

This section features screenshots of the Grafana node dashboard, which provides a detailed view of costs and resource utilization for nodes within the cluster.

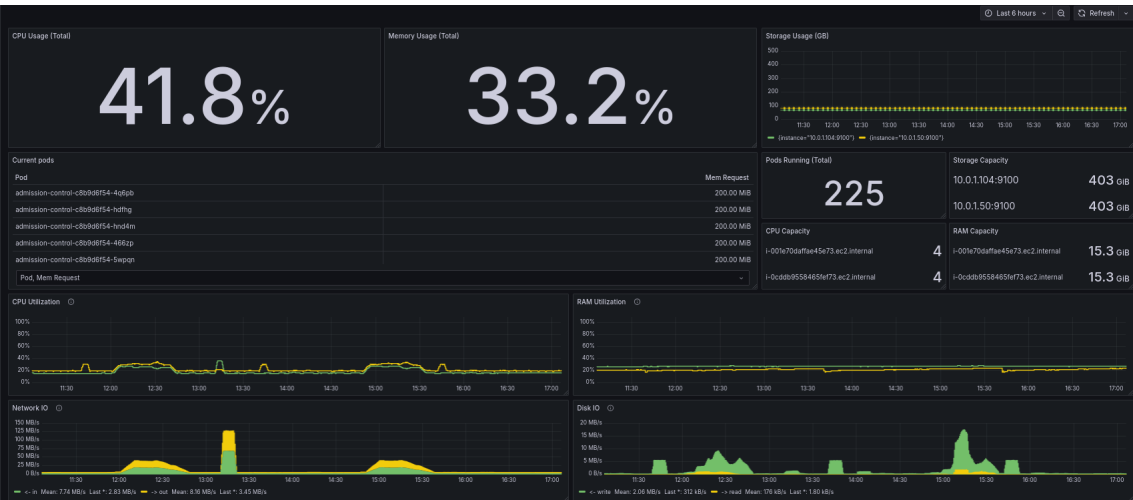


Figure C.10: Node Resource Utilization dashboard presenting CPU and memory usage percentages, detailed resource allocation metrics including the total number of pods running and storage capacity analysis, CPU utilization trends over time, memory usage patterns, and disk input/output performance monitoring across the cluster nodes.

C.1.4 Dashboards - Specific Scenarios

This section features screenshots of both dashboards for some specific scenarios, such as the removal of **PVCs** and Load Balancers. It illustrates how these actions impact the overall cost and resource utilization within the cluster, and how the framework can be used to monitor and analyze these changes effectively.

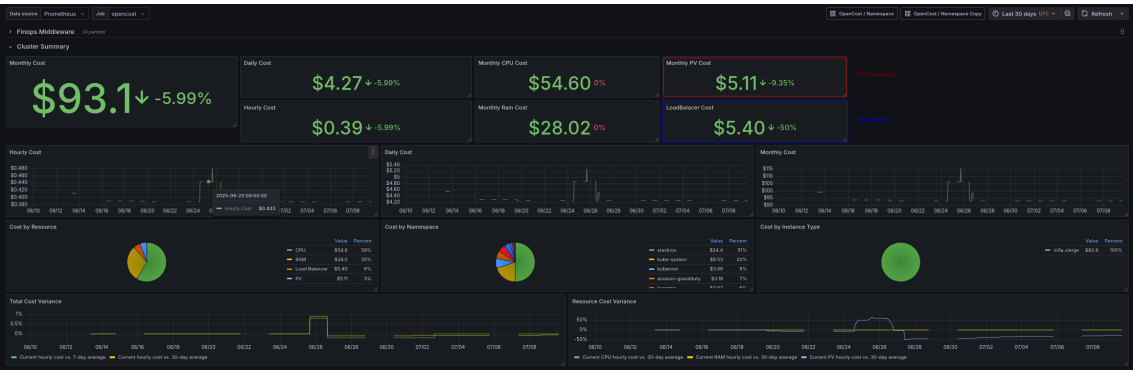


Figure C.11: Cluster Summary dashboard showing the impact of creating a Load Balancer, and removing a Persistent Volume on the cluster's cost. The impact can be seen in the specific dashboard for Persistent Volume and Load Balancer costs, hourly, daily, and monthly cost breakdowns, as well as the total cost and resource variance panels.

C.2 CloudWatch-based Framework



Figure C.12: EC2 Cost Analysis dashboard presenting total monthly and daily costs, potential savings opportunities, existing instance type distribution overview, and cost distribution analysis across different EC2 instance types.

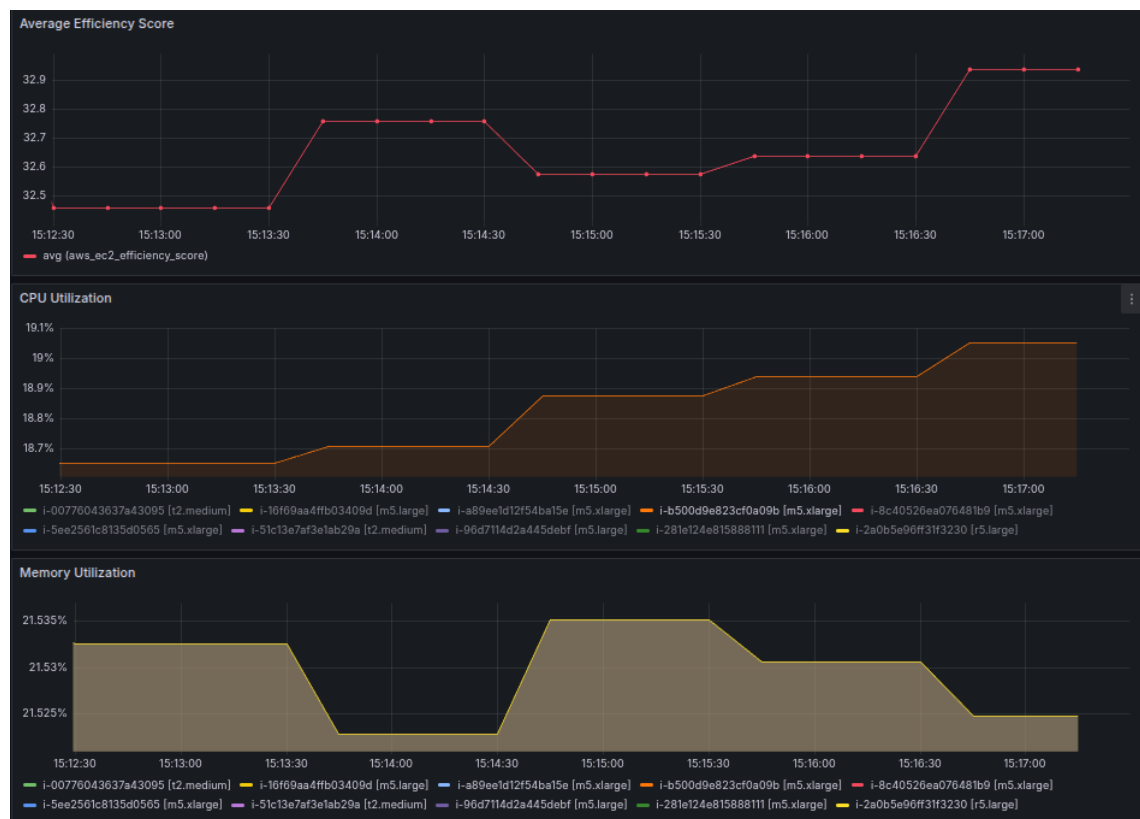


Figure C.13: EC2 Resource Utilization dashboard presenting average efficiency scores over time, and CPU and Memory utilization patterns across individual EC2 instances.

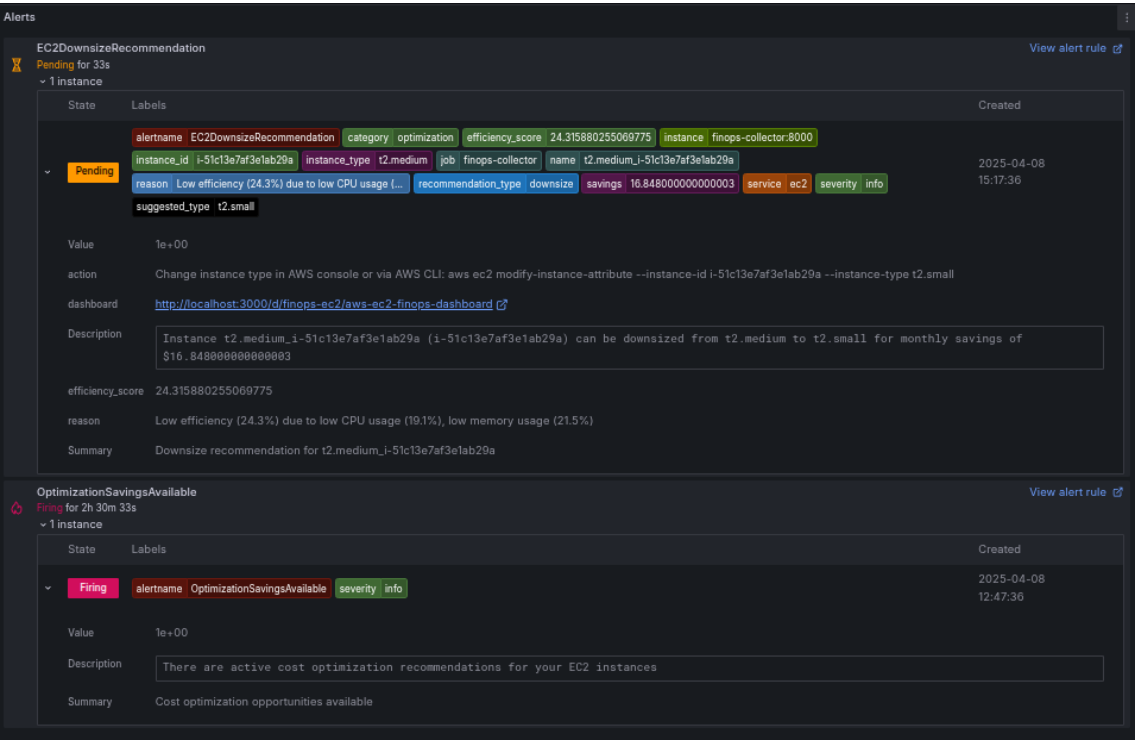


Figure C.14: EC2 Alerting dashboard presenting active alerts for EC2 instances. This dashboard serves more as a developer experience improvement than a feature, since Grafana already provides a dedicated alert tab.

Appendix D

Kubernetes-based Framework Monitoring Metrics

This appendix provides a comprehensive overview of the metrics used by the Kubernetes-based Framework and their sources.

Table D.1: Node Exporter Metrics

Metric	Description
instance:node_cpu_utilisation	CPU utilization percentage per node.
node_filesystem_size_bytes	Total size (capacity) of each mounted filesystem on the node.
node_memory_MemAvailable_bytes	Amount of memory available for new processes, including file cache.
node_memory_MemTotal_bytes	Total physical memory available on the node.

Table D.2: cAdvisor/Kubelet Metrics

Metric	Description
container_memory_usage_bytes	Total memory used by a container, including cache.
container_network_receive_bytes_total	Total number of bytes received by the container over the network since it started.
container_network_transmit_bytes_total	Total number of bytes sent by the container over the network since it started.
container_fs_writes_bytes_total	Total number of bytes written to the filesystem by the container.
container_fs_reads_bytes_total	Total number of bytes read from the filesystem by the container.
container_cpu_usage_seconds_total	CPU used over time, in seconds.
container_memory_working_set_bytes	Actual memory in use (excluding reclaimable cache).

Table D.3: OpenCost Metrics

Metric	Description
container_cpu_allocation	CPU allocated to a container (based on resource requests/limits).
container_memory_allocation_bytes	Memory allocated to the container (based on resource requests/limits).
kubecost_load_balancer_cost	Estimated cost associated with Load Balancers.
node_ram_hourly_cost	Hourly cost of RAM per node.
node_cpu_hourly_cost	Hourly cost of CPU per node.
node_total_hourly_cost	Total node hourly cost (CPU + RAM + other).
pvc_hourly_cost	Estimated hourly cost for Persistent Volumes.

Table D.4: kube-state-metrics Metrics

Metric	Description
kube_pod_container_resource_requests	CPU/memory requests per container.
kube_pod_container_resource_limits	CPU/memory limits per container.
kube_persistentvolume_capacity_bytes	Total capacity of a Persistent Volume.
kube_persistentvolumeclaim_info	PVC metadata: namespace, storage class, etc.
kube_persistentvolume_claim_ref	Relationship between PVC and the pod that uses it.

Appendix E

Alert Configurations

This appendix documents the AlertManager alert rules implemented for both the Kubernetes-based and CloudWatch-based Frameworks, providing explanations of their purpose and configuration.

The values in the alert rules below are the ones that were used in this dissertation, and align with the resource definitions and limits set in Section 5.2.5. Having said this, these values can and should be adjusted to fit the specific needs of each organization, as they may vary depending on the workload, resource utilization patterns, and cost optimization goals.

E.1 Kubernetes-based Framework

For a clearer understanding of the PromQL expressions used in these alert rules, it is recommended to refer to Appendix D, which provides a comprehensive overview of the metrics used by the Kubernetes-based Framework and their sources.

E.1.1 Cost-Related Alerts

Listing E.1: Low Cost Efficiency Alert

```
1 - alert: LowCostEfficiencyWarning
2   expr: avg(finops_efficiency_score{}) by (exported_namespace) < 50 and avg(
3     finops_efficiency_score{}) by (exported_namespace) >= 30
4   for: 24h
5   labels:
6     severity: warning
7     category: finops
8   annotations:
9     summary: "Low cost efficiency in namespace {{ $labels.exported_namespace }}"
```

```

9      description: "The cost efficiency score for namespace {{ $labels.
        exported_namespace }} is {{ $value | printf \"%.2f\" }}%, which is between
        30% and 50% for the last 24 hours. Requests for resources in this namespace
        are likely set too high. Check requests for resources in this namespace
        for possible inefficiencies."

```

This alert identifies namespaces with suboptimal cost efficiency scores. The expression checks if the average cost efficiency score is below 50% but above 30%, indicating that resources are not being utilized effectively relative to their costs. The 24-hour evaluation period helps filter out temporary spikes in resource utilization, allowing teams to focus on persistent inefficiencies that may require action.

Due to the nature of this metric, which is calculated based on the ratio of actual resource utilization to requested resources, resources are likely set to higher values than necessary, leading to wasted costs.

Listing E.2: Critical Low Cost Efficiency Alert

```

1 - alert: CriticalCostEfficiencyWarning
2   expr: avg(finops_efficiency_score{}) by (exported_namespace) < 30
3   for: 24h
4   labels:
5     severity: critical
6     category: finops
7   annotations:
8     summary: "Critical low cost efficiency in namespace {{ $labels.
        exported_namespace }}"
9     description: "The cost efficiency score for namespace {{ $labels.
        exported_namespace }} is {{ $value | printf \"%.2f\" }}%, which is below
        the critical threshold of 30% for the last 24 hours. Requests for resources
        in this namespace are likely set too high."

```

This alert identifies namespaces with very low cost efficiency scores, indicating that resources are being severely underutilized or overprovisioned. The expression is similar to the previous one, but with a lower threshold of 30% and a longer evaluation period of 24 hours.

Listing E.3: Daily Cost Increase Alert

```

1 - alert: DailyCostIncrease
2   expr: |
3     sum(
4       sum(avg_over_time(container_memory_allocation_bytes{job="opencost"}[24h])) by
         (namespace, instance)
5     * on(instance) group_left()
6       (node_ram_hourly_cost{job="opencost"} / (1024 * 1024 * 1024))

```

```

7      +
8      sum(avg_over_time(container_cpu_allocation{job="opencost"}[24h])) by (
9          namespace, instance)
10     * on(instance) group_left()
11     (node_cpu_hourly_cost{job="opencost"})
12 ) by (namespace)
13 >
14 (
15     sum(
16         sum(avg_over_time(container_memory_allocation_bytes{job="opencost"}[24h]
17             offset 1d)) by (namespace, instance)
18         * on(instance) group_left()
19         (node_ram_hourly_cost{job="opencost"} offset 1d / (1024 * 1024 * 1024))
20     +
21     sum(avg_over_time(container_cpu_allocation{job="opencost"}[24h] offset 1d))
22         by (namespace, instance)
23     * on(instance) group_left()
24     (node_cpu_hourly_cost{job="opencost"} offset 1d)
25 ) by (namespace)
26 ) * 1.5
27 for: 2h
28 labels:
29     severity: warning
30     category: finops
31 annotations:
32     summary: "Namespace {{ $labels.namespace }} cost increased by >50%"
33     description: "The compute cost for namespace {{ $labels.namespace }} has
34         increased by more than 50% compared to the previous day."

```

This alert detects daily cost increases by comparing the current day's total compute costs — the sum of memory and CPU costs — to the previous day's total compute costs. The alert is triggered when the current day's costs exceed the previous day's costs by more than 50%. The 2-hour evaluation period helps filter out short-term fluctuations in resource compute costs.

Listing E.4: Weekly Cost Increase Alert

```

1 - alert: WeeklyCostIncrease
2   expr: |
3     sum(
4       sum(avg_over_time(container_memory_allocation_bytes{job="opencost"}[7d])) by
5         (namespace, instance)
6       * on(instance) group_left()
7       (node_ram_hourly_cost{job="opencost"} / (1024 * 1024 * 1024))
8     +
9     sum(avg_over_time(container_cpu_allocation{job="opencost"}[7d])) by (
10       namespace, instance)
11     * on(instance) group_left()

```

```

10     (node_cpu_hourly_cost{job="opencost"})
11   ) by (namespace)
12   >
13   (
14     sum(
15       sum(avg_over_time(container_memory_allocation_bytes{job="opencost"}[7d]
16         offset 7d)) by (namespace, instance)
17       * on(instance) group_left()
18         (node_ram_hourly_cost{job="opencost"} offset 7d / (1024 * 1024 * 1024))
19     +
20     sum(avg_over_time(container_cpu_allocation{job="opencost"}[7d] offset 7d))
21       by (namespace, instance)
22     * on(instance) group_left()
23       (node_cpu_hourly_cost{job="opencost"} offset 7d)
24   ) by (namespace)
25   ) * 1.3
26 for: 2h
27 labels:
28   severity: warning
29   category: finops
30 annotations:
31   summary: "Namespace {{ $labels.namespace }} cost increased by >30%"
32   description: "The compute cost for namespace {{ $labels.namespace }} has
33     increased by more than 50% compared to the previous week."

```

This alert is similar to the previous one but detects weekly cost increases instead of daily ones. The smaller 30% threshold ensures that the alert is triggered for significant cost increases that may not be immediately visible in daily fluctuations.

Listing E.5: Cost Anomaly Alert

```

1 - alert: CostAnomalyWarning
2   expr: avg(finops_anomaly_score{}) by (exported_namespace) > 50 and avg(
3     finops_anomaly_score{}) by (exported_namespace) < 90
4   for: 2h
5   labels:
6     severity: warning
7     category: finops
8   annotations:
9     summary: "Cost anomaly detected in namespace {{ $labels.exported_namespace }}"
10    description: "A cost anomaly with score {{ $value | printf \"%.2f\" }}/100 has
11      been detected in namespace {{ $labels.exported_namespace }}. This may
12      indicate an unexpected cost increase."

```

This alert detects moderate cost anomalies that deviate from standard spending patterns. The score range of 50% to 90% indicates unusual cost variations. The 2-hour evaluation period helps

filter out short-term fluctuations while catching significant anomalies that may require investigation.

Listing E.6: Severe Cost Anomaly Alert

```
1 - alert: SevereCostAnomaly
2   expr: avg(finops_anomaly_score{}) by (exported_namespace) >= 90
3   for: 1h
4   labels:
5     severity: critical
6     category: finops
7   annotations:
8     summary: "Severe cost anomaly detected in namespace {{ $labels.
9       exported_namespace }}"
10    description: "A severe cost anomaly with score {{ $value | printf \"%.2f\"
11      }}/100 has been detected in namespace {{ $labels.exported_namespace }}.
12      Immediate investigation is recommended."
```

This alert identifies severe cost anomalies that require immediate attention. The expression is similar to the previous one, but with a higher threshold of 90 and a shorter evaluation period of 1 hour.

E.1.2 Optimization Recommendation Alerts

Listing E.7: High Savings Opportunity Alert

```
1 - alert: HighSavingsOpportunity
2   expr: sum(finops_optimization_savings{}) by (exported_namespace) > 100
3   for: 7d
4   labels:
5     severity: info
6     category: finops
7   annotations:
8     summary: "High potential savings identified for namespace {{ $labels.
9       exported_namespace }}"
10    description: "There is a potential for ${{ $value | printf \"%.2f\" }} monthly
11      savings in namespace {{ $labels.exported_namespace }}. Review the
12      recommendations in the FinOps dashboard."
```

This alert identifies namespaces with significant potential savings from optimization recommendations. The expression checks if the total optimization savings for a namespace exceed 100\$. This value can and should be adjusted based on the organization's cost optimization goals. The 7-day evaluation period helps filter out short-term fluctuations and ensures that the savings opportunity is persistent.

Listing E.8: Optimization Savings Available Alert

```
1 - alert: OptimizationSavingsAvailable
2   expr: count(finops_optimization_savings{} > 0) by (recommendation_type) > 0
3   for: 1h
4   labels:
5     severity: warning
6     category: finops
7   annotations:
8     summary: "Cost optimization opportunities available"
9     description: "There are {{ $value }} active '{{ $labels.recommendation_type }}'
      recommendations that have been pending for over 14 days."
```

This alert identifies when there are any active and pending cost optimization recommendations available for a namespace. It serves as a reminder to review and act on these recommendations to optimize costs and resource utilization.

E.1.3 Resource-Level Alerts

Listing E.9: Cluster CPU Underutilization Alert

```
1 - alert: ClusterCPUUnderutilization
2   expr: sum(rate(container_cpu_usage_seconds_total{container!=""}[1d])) / sum(
      kube_node_status_capacity{resource="cpu"}) < 0.3
3   for: 7d
4   labels:
5     severity: warning
6     category: finops
7   annotations:
8     summary: "Cluster underutilized"
9     description: "Cluster CPU utilization has been below 30% for 7 days. Consider
      rightsizing the cluster."
```

This alert monitors overall CPU utilization and identifies when the entire cluster is consistently underutilized. The 7-day evaluation period ensures this is not temporary low usage, but rather a trend that may indicate the cluster is larger than necessary.

Listing E.10: High CPU Waste Alert

```
1 - alert: HighCPUWaste
2   expr: finops_resource_waste{resource_type="cpu"} > 70
3   for: 24h
4   labels:
5     severity: warning
6     category: finops
```

```

7 annotations:
8   summary: "High CPU waste in namespace {{ $labels.exported_namespace }}"
9   description: "The CPU resources in namespace {{ $labels.exported_namespace }}
    are {{ $value | printf \"%.2f\\\" }}% wasted over the last 24 hours (
    requested but not used)."
```

This alert identifies namespaces with consistently low CPU utilization relative to requested resources. The 70% threshold indicates that less than 30% of the requested CPU is being used, suggesting that resources may be overprovisioned. The 24-hour evaluation period helps filter out false-positive alerts caused by short-term fluctuations in resource utilization.

Listing E.11: High Memory Waste Alert

```

1 - alert: HighMemoryWaste
2   expr: finops_resource_waste{resource_type="memory"} > 70
3   for: 24h
4   labels:
5     severity: warning
6     category: finops
7   annotations:
8     summary: "High memory waste in namespace {{ $labels.exported_namespace }}"
9     description: "The memory waste in namespace {{ $labels.exported_namespace }} is
    {{ $value | printf \"%.2f\\\" }}%, which is above the threshold of 70% for
    the last 24 hours."
```

This alert identifies namespaces with consistently low memory utilization relative to requested resources. The 70% threshold indicates that less than 30% of requested memory is being used, suggesting that resources may be overprovisioned. The 24-hour evaluation period helps filter out false-positive alerts caused by short-term fluctuations in resource utilization.

Listing E.12: Idle Persistent Volume Alert

```

1 - alert: IdlePersistentVolumes
2   expr: kube_persistentvolume_status_phase{phase="Available"} == 1
3   for: 7d
4   labels:
5     severity: warning
6     category: finops
7   annotations:
8     summary: "Idle persistent volume {{ $labels.persistentvolume }}"
9     description: "Persistent volume {{ $labels.persistentvolume }} has been
    available (unused) for 7 days."
```

This alert identifies **PVs** that have been available — created, but not bound to any **PVC** —

for an extended period of time, indicating that they may be unused and incurring costs without providing value.

Listing E.13: Idle Load Balancer Alert

```
1  - alert: IdleLoadBalancer
2    expr: |
3      sum(kube_service_spec_type{type="LoadBalancer"}) by (namespace, service)
4      unless sum(kube_pod_info) by (namespace, service)
5    for: 7d
6    labels:
7      severity: warning
8      category: finops
9    annotations:
10     summary: "Idle LoadBalancer service in namespace {{ $labels.namespace }}"
11     description: "LoadBalancer service {{ $labels.service }} in namespace {{
        $labels.namespace }} appears to have no backend pods for 7 days."
```

This alert identifies Load Balancer services that have no backend pods — i.e., no pods associated with the service — for an extended period of time, indicating that they may be unused and incurring costs without providing value.

Listing E.14: Missing Memory Limits Alert

```
1  - alert: MissingMemoryLimits
2    expr: |
3      finops_optimization_savings{recommendation_type="add_memory_limits", namespace=
4        "finops"} = 0
5    for: 5m
6    labels:
7      severity: warning
8      team: finops
9    annotations:
10     summary: "Missing memory limits in namespace {{ $labels.exported_namespace }}"
11     description: |
12       Pod {{ $labels.pod }} in namespace {{ $labels.exported_namespace }} is
13       missing memory resource limits.
```

This alert identifies pods that are missing memory limits, which means they are not correctly configured and may lead to inefficient resource utilization and the consumption of resources without proper control.

Listing E.15: Missing CPU Limits Alert

```

1 - alert: MissingCPULimits
2   expr: |
3     finops_optimization_savings{recommendation_type="add_cpu_limits", namespace="
4       finops"} = 0
5   for: 5m
6   labels:
7     severity: warning
7     team: finops
8   annotations:
9     summary: "Missing CPU limits in namespace {{ $labels.exported_namespace }}"
10    description: |
11      Pod {{ $labels.pod }} in namespace {{ $labels.exported_namespace }} is
        missing CPU resource limits.

```

This alert identifies pods that are missing CPU limits, which means they are not correctly configured and may lead to inefficient resource utilization and the consumption of resources without proper control.

Listing E.16: Missing Memory Requests and Limits Alert

```

1 - alert: MissingMemoryRequestsAndLimits
2   expr: |
3     finops_optimization_savings{recommendation_type="add_memory_limits_and_requests
4       ", namespace="finops"} = 0
5   for: 5m
6   labels:
7     severity: warning
7     team: finops
8   annotations:
9     summary: "Missing memory requests and limits in namespace {{ $labels.
10       exported_namespace }}"
11    description: |
        Pod {{ $labels.pod }} in namespace {{ $labels.exported_namespace }} is
        missing memory resource requests and limits. Recommendation generated by
        FinOps API.

```

This alert identifies pods missing memory requests and limits, which means they are not correctly configured. It may lead to cases where pods are not guaranteed to have the necessary memory resources available, leading to potential performance issues or crashes.

Listing E.17: Missing CPU Requests and Limits Alert

```

1 - alert: MissingCPURequestsAndLimits
2   expr: |

```

```

3     finops_optimization_savings{recommendation_type="add_cpu_limits_and_requests",
      namespace="finops"} = 0
4   for: 5m
5   labels:
6     severity: warning
7     team: finops
8   annotations:
9     summary: "Missing CPU requests and limits in namespace {{ $labels.
      exported_namespace }}"
10    description: |
11      Pod {{ $labels.pod }} in namespace {{ $labels.exported_namespace }} is
      missing CPU resource requests and limits. Recommendation generated by
      FinOps API.

```

This alert identifies pods missing CPU requests and limits, which means they are not correctly configured. It may lead to cases where pods are not guaranteed to have the necessary CPU resources available, leading to potential performance issues or crashes.

E.2 CloudWatch-based Framework

E.2.1 Cost-Related Alerts

Listing E.18: Cost Spike Alert

```

1 - alert: CostSpikeWarning
2   expr: rate(aws_ec2_monthly_cost{type='total'}[10m]) > 0.2
3   for: 5m
4   labels:
5     severity: warning
6     service: ec2
7     category: cost
8   annotations:
9     summary: 'Cost spike detected in EC2 spending'
10    description: 'EC2 costs have spiked by more than 20% in the last 10 minutes.'

```

This alert serves as an early warning for cost increases. It monitors the rate of change in total EC2 costs over a 10-minute window and triggers when costs increase by more than 20%. The 5-minute evaluation period helps filter short-term fluctuations and false positives while accounting for the alert's real-time nature.

Listing E.19: Critical Cost Spike Alert

```

1 - alert: CostSpikeCritical
2   expr: rate(aws_ec2_monthly_cost{type='total'}[5m]) > 0.5

```

```
3 for: 2m
4 labels:
5     severity: critical
6     service: ec2
7     category: cost
8 annotations:
9     summary: 'CRITICAL cost spike detected in EC2 spending'
10    description: 'EC2 costs have spiked by over 50% in the last 5 minutes.
        Immediate investigation required!'
```

This alert indicates cost anomalies that require immediate attention. The 50% threshold and shorter 2-minute evaluation period are designed to catch these anomalies quickly, so that teams can investigate and act on them before they lead to significant overspending.

Listing E.20: Week-over-Week Cost Increase Alert

```
1 - alert: CostIncrease
2 expr: |
3     (
4         sum(aws_ec2_monthly_cost{type='total'})
5         /
6         sum(aws_ec2_monthly_cost{type='total'} offset 7d)
7     ) > 1.2
8 labels:
9     severity: warning
10 annotations:
11     summary: 'Significant cost increase detected'
12     description: 'EC2 costs have increased by more than 20% compared to last week'
```

This alert detects weekly cost increases by comparing the current month's total EC2 costs to the previous week's calculated monthly costs. This may help teams identify issues that are not immediately visible or that may be related to longer-term trends, such as new deployments or changes in usage patterns.

E.2.2 Resource Utilization and Efficiency Alerts

Listing E.21: Low CPU Utilization Alert

```
1 - alert: LowCPUUtilizationWarning
2 expr: avg_over_time(aws_ec2_cpuutilization[30m]) < 20
3 for: 15m
4 labels:
5     severity: warning
6     service: ec2
```

```

7     category: utilization
8     instance_id: '{{ $labels.instance_id }}'
9     instance_type: '{{ $labels.instance_type }}'
10  annotations:
11     summary: 'Very low CPU utilization on {{ $labels.name }}'
12     description: 'Instance {{ $labels.name }} ({{ $labels.instance_id }}) has CPU
        utilization below 20% for the last 15 minutes. Consider rightsizing or
        terminating if this pattern continues.'
13     instance_details: 'Type: {{ $labels.instance_type }}'

```

This alert identifies instances with consistently low CPU utilization that may be candidates for downsizing or termination. The 20% threshold and 15-minute evaluation period help filter out short-term spikes or fluctuations, while the 30-minute average provides a more stable view of utilization trends.

Listing E.22: Completely Idle Instance Alert

```

1 - alert: CompletelyIdleInstance
2 expr: aws_ec2_cpuutilization < 5 and aws_ec2_memoryutilization < 10 and
        aws_ec2_networkin < 15000 and aws_ec2_networkout < 20000
3 for: 30m
4 labels:
5     severity: critical
6     service: ec2
7     category: idle
8     instance_id: '{{ $labels.instance_id }}'
9     instance_type: '{{ $labels.instance_type }}'
10 annotations:
11     summary: 'Instance {{ $labels.name }} appears completely idle'
12     description: 'Instance {{ $labels.name }} ({{ $labels.instance_id }}) shows
        virtually no activity for 30 minutes. This instance may be forgotten or
        unnecessary and generates costs without providing value.'
13     suggested_action: 'Investigate if this instance is needed or can be terminated.
        Estimated monthly savings: ${{ $labels.savings }}'
14     instance_details: 'Type: {{ $labels.instance_type }}'

```

This alert identifies instances with virtually no activity, indicating they may be deleted or downsized. This alert checks CPU, memory, and network utilization metrics. The CPU and memory thresholds are set to 5% and 10%, being these the values that we defined in Section 5.2.5, and the network thresholds of 50KB/s inbound and 20KB/s outbound correspond to minimal keepalive traffic and system monitoring overhead. The 30-minute evaluation periods filter typical instance initialization times and short-term fluctuations while still catching consistently idle instances.

Listing E.23: EC2 Downsize Recommendation Alert

```

1 - alert: EC2DownsizeRecommendation
2 expr: aws_ec2_recommendation_info{recommendation_type='downsize'}
3 for: 1m
4 labels:
5     severity: info
6     service: ec2
7     category: optimization
8     instance_id: '{{ $labels.instance_id }}'
9     instance_type: '{{ $labels.instance_type }}'
10 annotations:
11     summary: 'Downsize recommendation for {{ $labels.name }}'
12     description: 'Instance {{ $labels.name }} ({{ $labels.instance_id }}) can be
13         downsized from {{ $labels.instance_type }} to {{ $labels.suggested_type }}
14         for monthly savings of ${{ $labels.savings }}'
15     efficiency_score: '{{ $labels.efficiency_score }}'
16     reason: '{{ $labels.reason }}'
17     dashboard: 'http://localhost:3000/d/finops-ec2/aws-ec2-finops-dashboard'
18     action: 'Change instance type in AWS console or via AWS CLI: aws ec2 modify-
19         instance-attribute --instance-id {{ $labels.instance_id }} --instance-type
20         {{ $labels.suggested_type }}'

```

This alert is triggered when there are active recommendations for downsizing EC2 instances. It provides details about the instance, the suggested new type, and potential monthly savings.

E.2.3 Optimization Recommendation Alerts

Listing E.24: EC2 Terminate Recommendation Alert

```

1 - alert: EC2TerminateRecommendation
2 expr: aws_ec2_recommendation_info{recommendation_type='terminate'}
3 for: 1m
4 labels:
5     severity: warning
6     service: ec2
7     category: optimization
8     instance_id: '{{ $labels.instance_id }}'
9     instance_type: '{{ $labels.instance_type }}'
10 annotations:
11     summary: 'Terminate recommendation for {{ $labels.name }}'
12     description: 'Instance {{ $labels.name }} ({{ $labels.instance_id }}) appears
13         to be unused and could be terminated for monthly savings of ${{ $labels.
14         savings }}'
15     efficiency_score: '{{ $labels.efficiency_score }}'
16     reason: '{{ $labels.reason }}'
17     dashboard: 'http://localhost:3000/d/finops-ec2/aws-ec2-finops-dashboard'

```

```
16     action: 'Terminate instance in AWS console or via AWS CLI: aws ec2 terminate-
        instances --instance-ids {{ $labels.instance_id }}'
```

This alert is triggered when there are active recommendations for terminating EC2 instances. It provides details about the instance, the suggested new type, and potential monthly savings.

Listing E.25: Optimization Savings Available Alert

```
1 - alert: OptimizationSavingsAvailable
2 expr: |
3     count(aws_ec2_recommendation_info) > 0
4 for: 1h
5 labels:
6     severity: info
7 annotations:
8     summary: 'Cost optimization opportunities available'
9     description: 'There are active cost optimization recommendations for your EC2
        instances'
```

This alert is triggered when any active cost optimization recommendations are available for EC2 instances. It serves as a reminder to review and act on these recommendations to optimize costs and resource utilization.

E.2.4 Trend Analysis Alerts

Listing E.26: Declining Efficiency Score Alert

```
1 - alert: DecliningEfficiencyScore
2 expr: rate(aws_ec2_efficiency_score[1h]) < -5
3 for: 2h
4 labels:
5     severity: info
6     service: ec2
7     category: trend
8     instance_id: '{{ $labels.resource_id }}'
9 annotations:
10    summary: 'Declining efficiency for {{ $labels.name }}'
11    description: 'Instance {{ $labels.name }} ({{ $labels.instance_id }}) is
        showing a declining efficiency trend over the past hour. Current score: {{
        $labels.efficiency_score }}%'
12    suggested_action: 'Monitor this instance for potential rightsizing
        opportunities if the trend continues.'
```

This alert identifies instances with declining efficiency scores exceeding 5 points per hour. It provides insights into the instance's overall efficiency and suggests further monitoring for potential rightsizing opportunities.

Appendix F

User Demand Survey

F.1 Survey Questions

Evaluating FinOps Frameworks for Cloud Cost Optimization

As part of my thesis *Automated FinOps for Cloud Infrastructure Cost Management*, I've developed two frameworks for cloud cost optimization for both AWS EC2 and Kubernetes. These frameworks focus on real-time cost monitoring, observability, actionable insights (e.g., cost anomaly detection, idle resource detection, and others), and recommendations tailored to specific use cases.

These frameworks were developed to address the limitation in AWS Cost Explorer, where cost data can be delayed by up to 24 hours, potentially causing cloud tenants to incur unexpected bills due to cost spikes, misconfigurations, and other issues.

Although these frameworks are distinct in their technical focus, they share a unified goal and are ultimately components of a single integrated solution for comprehensive cloud cost management.

Note: This form requires you to be logged in to make sure there are no duplicate answers. However, your email address will not be collected or stored.

* Indica uma pergunta obrigatória

1. What is your primary role? *

Marcar apenas uma oval.

- ☐ DevOps/Platform Engineer
- ☐ Site Reliability Engineer (SRE)
- ☐ Software Developer/Engineer
- ☐ Engineering Manager/Team Lead
- ☐ Outra: _____

2. What is the approximate size of your organization? *

Marcar apenas uma oval.

- ☐ 1-9 employees
- ☐ 10-49 employees
- ☐ 50-249 employees
- ☐ 250+ employees

3. Have you ever worked with any cloud platform? *

Marcar apenas uma oval.

- ☐ Yes, I currently work with cloud platforms
- ☐ Yes, I have worked with cloud platforms in the past
- ☐ No, I have never worked with cloud platforms

4. To what extent do cloud costs affect your decisions? *

Marcar apenas uma oval.

- ☐ Cloud costs play a significant role in my decision-making
- ☐ I consider cloud costs in some of my decisions
- ☐ I am aware of cloud costs, but they rarely influence my decisions
- ☐ I do not care about cloud costs

Current Cloud Cost Management

In this section, we'd like to understand the current state of your organization's cloud cost management.

5. How would you rate your current cloud cost visibility? *

Marcar apenas uma oval.

- ☐ Excellent - We have comprehensive visibility and control
- ☐ Good - We can see costs but lack detailed insights
- ☐ Fair - Basic cost tracking with limited analysis
- ☐ Poor - Limited visibility into cost drivers
- ☐ Very Poor - Little to no cost sources

6. What cloud cost challenges has your team faced? (Select all that apply) *

Marcar tudo o que for aplicável.

- ☐ Unexpected cost spikes
- ☐ Difficulty identifying cost sources
- ☐ Over-provisioned resources
- ☐ Unused or idle resources
- ☐ Manual effort required for cost analysis
- ☐ Delayed cost reporting by native cloud tools (e.g. AWS Cost Explorer)
- ☐ Difficulty forecasting future costs
- ☐ No standardized approach to cost optimization
- ☐ Outra: _____

7. How much time does your team currently spend on manual cloud cost analysis per month? *

Marcar apenas uma oval.

- ☐ None, we have cloud cost analysis fully automated
- ☐ None, our team is not concerned with cloud costs
- ☐ 1-5 hours
- ☐ 6-15 hours
- ☐ 16-30
- ☐ 30+ hours

8. What is your current process for identifying cost optimization opportunities? (Select * all that apply)

Marcar tudo o que for aplicável.

- ☐ Manual review of AWS Cost Explorer
- ☐ Monthly/quarterly cost reports
- ☐ AWS Trusted Advisor recommendations
- ☐ Third-party cost management tools
- ☐ Custom scripts or tools
- ☐ None
- ☐ Outra: _____

EC2 FinOps Framework

This framework is designed to optimize AWS EC2 costs by leveraging real-time monitoring and automated analytics. Key features include:

- Real-time cost observability by continuously fetching AWS CloudWatch metrics;
- Automated identification of cost-saving opportunities (e.g., rightsizing instances, terminating idle instances);
- Interactive Grafana dashboards for cost visibility;
- Data formatted according to Prometheus metrics standard;
- Proactive alerts for cost spikes and optimization insights;
- Modular architecture for easy extendability to other AWS services.

The framework currently focuses on EC2 instances, with plans to extend to other AWS services.

9. How would you rate the features of this framework? (1 = Not Important, 5 = Critical) *

Marcar apenas uma oval por linha.

	1	2	3	4	5
Prevent unexpected cost spikes and budget overruns	☐	☐	☐	☐	☐
Recommend rightsizing of over-provisioned instances	☐	☐	☐	☐	☐
Recommend idle and forgotten resource elimination	☐	☐	☐	☐	☐
Recommend the most suitable instance types	☐	☐	☐	☐	☐
Real-time efficiency scoring and recommendations	☐	☐	☐	☐	☐
Multiple operation modes (AWS/LocalStack/CSV) for different environments	☐	☐	☐	☐	☐
Automated alerts for optimization opportunities	☐	☐	☐	☐	☐

10. Have you ever managed or deployed applications on a Kubernetes environment? *

Marcar apenas uma oval.

☐ Yes

☐ No

Kubernetes FinOps Framework

This framework addresses cost management challenges in Kubernetes (K8S) environments. Built on top of OpenCost, it offers:

- Real-time cost observability through the usage of opencost;
- Alerts for cost anomalies and optimization opportunities;
- One-script deployment for AWS EKS clusters;
- Interactive Grafana dashboards for cost visibility;
- Data formatted according to Prometheus metrics standards;
- Kubernetes "flavor" agnostic.

11. How do you currently monitor Kubernetes costs? (Select all that apply) *

Marcar tudo o que for aplicável.

- ☐ AWS Cost Explorer
- ☐ Third-party tools (e.g., Kubecost, Datadog, etc.)
- ☐ Local monitoring (e.g., custom scripts/dashboards)
- ☐ We don't currently monitor kubernetes costs
- ☐ Outra: _____

12. What are your biggest pain points with current Kubernetes cost management? (Select top 3) *

Marcar tudo o que for aplicável.

- ☐ Lack of granular visibility over kubernetes costs
- ☐ No proactive alerts for cost spikes
- ☐ Time-consuming manual analysis
- ☐ Difficulty to identify optimization opportunities
- ☐ Complex setup and maintenance of monitoring tools
- ☐ Lack of real-time visibility
- ☐ Limited forecasting and budgeting open-source tools
- ☐ Outra: _____

13. Rate the importance of each feature for your organization (1 = Not Important, 5 = ★ Critical)

Marcar apenas uma oval por linha.

	1	2	3	4	5
Real-time cost dashboards per cluster, namespace, and node	☐	☐	☐	☐	☐
Cost anomaly detection and alerts	☐	☐	☐	☐	☐
Resource waste identification (CPU/Memory)	☐	☐	☐	☐	☐
Hourly, daily, and monthly cost forecasting	☐	☐	☐	☐	☐
Integration with Prometheus and Grafana	☐	☐	☐	☐	☐
"One-click" deployment on existing clusters	☐	☐	☐	☐	☐
Pre-built alerting rules for cost optimization	☐	☐	☐	☐	☐

14. Which cost optimization alerts would be most valuable? (Select top 3) *

Marcar tudo o que for aplicável.

- ☐ High resource waste (unused CPU/memory)
- ☐ Cost anomalies (e.g. unexpected cost spikes)
- ☐ Idle or underutilized resource
- ☐ Weekly cost increases
- ☐ Low cost-efficiency scores per namespace
- ☐ High potential savings opportunities
- ☐ Persistent optimization recommendations not acted upon
- ☐ Outra: _____

15. What cost visibility granularity is most important to you? *

Marcar tudo o que for aplicável.

- ☐ Cluster-level: Costs aggregated across the entire cluster
- ☐ Namespace-level: Costs grouped by namespace (e.g., dev, prod)
- ☐ Pod-level: Costs for specific workloads running in pods
- ☐ Container-level: Costs for individual containers within pods
- ☐ Node-level: Costs attributed to specific nodes (e.g., EC2 in AWS)
- ☐ Resource type costs: Aggregated costs by CPU, memory, storage

Implementation and adoption

This section focuses on possible concerns with the implementation/adoption of these frameworks.

16. How important is easy deployment/setup for adopting a new monitoring solution? *

Marcar apenas uma oval.

1 2 3 4 5

 Criti ☐ ☐ ☐ ☐ ☐ Not important

17. What would be the biggest barriers to adopting these frameworks (Select all that apply) *

Marcar tudo o que for aplicável.

- ☐ None
- ☐ Security/compliance concerns
- ☐ Resource overhead on clusters
- ☐ Time/resources spent to integrate and use the solutions
- ☐ Outra: _____

18. What would success look like for these frameworks? (Select top 3) *

Marcar tudo o que for aplicável.

- ☐ Overall reduction of costs
- ☐ Faster identification of cost optimization opportunities
- ☐ Reduced time spent on manual cost analysis
- ☐ Proactive prevention of cost anomalies
- ☐ Improved awareness and visibility of costs
- ☐ Outra: _____

19. How much cost reduction would justify implementing these frameworks? *

Marcar apenas uma oval.

- ☐ Any measurable reduction
- ☐ 5-10% cost reduction
- ☐ 10-20% cost reduction
- ☐ 20%+ cost reduction
- ☐ Cost reduction is not the primary goal

20. Beyond cost savings, what other benefits would be valuable? (Select all that apply) *

Marcar tudo o que for aplicável.

- ☐ Improved resource utilization efficiency
- ☐ Real-time observability over cloud costs and their sources
- ☐ Real-time cost predictions
- ☐ Improved awareness and visibility of costs
- ☐ Outra: _____

21. Based on the description, how likely would you be to use these frameworks? *

Marcar apenas uma oval.

- ☐ Very likely
- ☐ Likely
- ☐ Neutral
- ☐ Unlikely
- ☐ Very unlikely

22. If you have additional comments or suggestions, feel free to leave them here.

Este conteúdo não foi criado nem aprovado pela Google.

Google Formulários

F.2 Survey Results

F.2.1 Respondent Demographics and Background

This section presents the demographic information collected from all 21 survey respondents.

Table F.1: Survey respondent demographics across four key categories: primary role, organization size, cloud platform experience, and Kubernetes experience. The data shows a predominance of technical professionals from large enterprises with substantial cloud computing experience.

Category	Count	Percentage
Primary Role		
DevOps/Platform Engineer	8	38.1%
Software Developer/Engineer	6	28.6%
Engineering Manager/Team Lead	4	19.0%
Site Reliability Engineer	2	9.5%
FinOps Analyst	1	4.8%
Organization Size		
250+ employees	18	85.7%
50-249 employees	2	9.5%
10-49 employees	1	4.8%
Cloud Platform Experience		
Currently working with cloud platforms	19	90.5%
Previous cloud experience	2	9.5%
Kubernetes Experience		
Has Kubernetes experience	18	85.7%
No Kubernetes experience	3	14.3%

F.2.2 Current Cost Management Practices

The following tables present current cost management practices as reported by all 21 respondents.

Table F.2: Assessment of cloud cost impact on organizational decision-making and current cost visibility levels among survey respondents. The data reveals a significant gap between the importance of costs in decision-making (76.2% significant role) and comprehensive visibility capabilities (23.8% excellent visibility).

Response	Count	Percentage
Cloud Cost Impact on Decisions		
Significant role in decision-making	16	76.2%
Consider costs in some decisions	3	14.3%
Aware but rarely influences decisions	2	9.5%
Do not care about cloud costs	0	0.0%
Current Cloud Cost Visibility Rating		
Good - Can see costs but lacks detailed insights	8	38.1%
Fair - Basic cost tracking with limited analysis	8	38.1%
Excellent - Comprehensive visibility and control	5	23.8%
Poor - Limited visibility into cost drivers	0	0.0%
Very Poor - Little to no cost sources	0	0.0%

Table F.3: Distribution of monthly time investment in manual cloud cost analysis activities across surveyed organizations. Results indicate that 90.5% of organizations rely on manual processes, with 47.6% dedicating more than 6 hours monthly to these activities.

Response	Count	Percentage
Monthly Time Spent on Manual Cost Analysis		
1-5 hours	9	42.9%
16-30 hours	5	23.8%
6-15 hours	4	19.0%
30+ hours	1	4.8%
None - fully automated	1	4.8%
None - team not concerned with costs	1	4.8%

Table F.4: Primary cloud cost management challenges faced by surveyed organizations, ranked by prevalence. Unexpected cost spikes (81.0%) and manual analysis requirements (71.4%) emerge as the most significant operational pain points.

Challenge	Count	Percentage
Unexpected cost spikes	17	81.0%
Manual effort required for cost analysis	15	71.4%
Over-provisioned resources	14	66.7%
Unused or idle resources	13	61.9%
Delayed cost reporting by native cloud tools	12	57.1%
Difficulty identifying cost sources	11	52.4%
Difficulty forecasting future costs	10	47.6%
No standardized approach to cost optimization	9	42.6%

Table F.5: Methods currently employed by organizations to identify cloud cost optimization opportunities. Manual AWS Cost Explorer review dominates (90.5%), with limited adoption of automated or third-party solutions.

Process	Count	Percentage
Manual review of AWS Cost Explorer	19	90.5%
Monthly/quarterly cost reports	8	38.1%
AWS Trusted Advisor recommendations	4	19.0%
Third-party cost management tools	3	14.3%
Custom scripts or tools	3	14.3%
None	1	4.8%

F.2.3 CloudWatch-based Framework Evaluation

All 21 respondents evaluated the CloudWatch-based framework features, rating their importance on a 5-point scale.

Table F.6: Feature importance ratings for the CloudWatch-based EC2 framework on a 5-point scale (1=Not Important, 5=Very Important). Prevention of unexpected costs received the highest rating ($\mu = 4.905$), while multiple operation modes received the lowest ($\mu = 3.429$). High Ratings indicates the percentage of respondents rating the feature ≥ 4 .

Feature	Count	μ	σ	High Ratings
Prevent unexpected cost spikes and budget overruns	21	4.905	0.294	100.0%
Automated alerts for optimization opportunities	21	4.571	0.660	90.5%
Real-time efficiency scoring and recommendations	21	4.476	0.794	90.5%
Recommend rightsizing of over-provisioned instances	21	4.381	0.844	76.2%
Recommend idle and forgotten resource elimination	21	4.286	0.700	85.7%
Recommend the most suitable instance types	21	3.857	0.990	61.9%
Multiple operation modes AWS/LocalStack/CSV	21	3.429	1.003	38.1%

F.2.4 Kubernetes-based Framework Evaluation

The Kubernetes-specific questions were answered only by the 18 respondents who indicated having experience with Kubernetes. All percentages and calculations in this subsection are based on these 18 respondents.

Table F.7: Current approaches to Kubernetes cost monitoring among organizations with container experience. AWS Cost Explorer remains the dominant tool (83.3%) despite its limitations regarding real-time visibility and granularity in Kubernetes environments.

Monitoring Approach	Count	Percentage
AWS Cost Explorer	15	83.3%
Third-party tools (Kubecost, Datadog, etc.)	6	33.3%
Local monitoring (custom scripts/dashboards)	3	16.7%
We don't currently monitor Kubernetes costs	1	5.6%

Table F.8: Specific pain points in Kubernetes cost management identified by respondents with container experience. Lack of real-time and granular visibility both affect 77.8% of organizations, representing the most critical gaps in current tooling.

Pain Point	Count	Percentage
Lack of real-time visibility	14	77.8%
Lack of granular visibility over Kubernetes costs	14	77.8%
Time-consuming manual analysis	9	50.0%
No proactive alerts for cost spikes	7	38.9%
Complex setup and maintenance of monitoring tools	5	27.8%
Difficulty to identify optimization opportunities	3	16.7%
Limited forecasting and budgeting open-source tools	2	11.1%

Table F.9: Feature importance ratings for the Kubernetes-based framework on a 5-point scale (1=Not Important, 5=Very Important). Cost anomaly detection received unanimous high ratings (100%), while deployment ease and forecasting capabilities received more moderate assessments. High Ratings indicates the percentage of respondents rating the feature ≥ 4 .

Feature	Count	μ	σ	High Ratings
Cost anomaly detection and alerts	18	4.778	0.416	100.0%
Real-time cost dashboards per cluster, namespace, and node	18	4.667	0.745	94.4%
Resource waste identification (CPU/Memory)	18	4.444	0.685	88.9%
Integration with Prometheus and Grafana	18	4.444	0.831	88.9%
Pre-built alerting rules for cost optimization	18	4.389	0.826	77.8%
Hourly, daily, and monthly cost forecasting	18	3.778	0.853	50.0%
One-click deployment on existing clusters	18	3.611	1.297	50.0%

Table F.10: Priority ranking of cost optimization alert types for Kubernetes environments. Cost anomalies (88.9%) and weekly cost increases (66.7%) are considered most valuable for proactive cost management.

Alert Type	Count	Percentage
Cost anomalies (e.g. unexpected cost spikes)	16	88.9%
Weekly cost increases	12	66.7%
High potential savings opportunities	10	55.6%
High resource waste (unused CPU/memory)	9	50.0%
Idle or underutilized resource	4	22.2%
Low cost-efficiency scores per namespace	2	11.1%
Persistent optimization recommendations not acted upon	1	5.6%

Table F.11: Preferred levels of cost visibility granularity for Kubernetes cost management. Cluster-level (72.2%) and namespace-level (66.7%) aggregations are most valued, aligning with typical organizational boundaries.

Granularity Level	Count	Percentage
Cluster-level: Costs aggregated across entire cluster	13	72.2%
Namespace-level: Costs grouped by namespace	12	66.7%
Pod-level: Costs for specific workloads running in pods	9	50.0%
Node-level: Costs attributed to specific nodes	9	50.0%
Resource type costs: Aggregated by CPU, memory, storage	9	50.0%
Container-level: Costs for individual containers	8	44.4%

F.2.5 Implementation and Adoption Factors

The questions in this section were answered only by the 18 respondents with Kubernetes experience, as they are related to implementation considerations for both frameworks.

Table F.12: Assessment of deployment ease importance and identified implementation barriers for cloud cost management frameworks. Security/compliance concerns represent the primary barrier (61.1%), while deployment ease shows mixed importance ratings.

Category	Count	Percentage
Easy Deployment Importance (5-point scale)		
Critical (5)	8	44.4%
Important (4)	4	22.2%
Neutral (3)	1	5.6%
Less Important (2)	2	11.1%
Not Important (1)	3	16.7%
Implementation Barriers		
Security/compliance concerns	11	61.1%
Resource overhead on clusters	8	44.4%
Time/resources spent to integrate and use solutions	7	38.9%
None	5	27.8%

Table F.13: Organizational success criteria for cloud cost management frameworks and minimum cost reduction thresholds that justify implementation. Overall cost reduction (83.3%) dominates success metrics, while 72.2% of organizations would implement the frameworks for any measurable cost savings.

Criterion/Expectation	Count	Percentage
Success Criteria		
Overall reduction of costs	15	83.3%
Faster identification of cost optimization opportunities	11	61.1%
Reduced time spent on manual cost analysis	10	55.6%
Proactive prevention of cost anomalies	9	50.0%
Improved awareness and visibility of costs	9	50.0%
Cost Reduction Justification Threshold		
Any measurable reduction	13	72.2%
10-20% cost reduction	4	22.2%
5-10% cost reduction	1	5.6%
20%+ cost reduction	0	0.0%
Cost reduction is not the primary goal	0	0.0%

Table F.14: Expected benefits beyond direct cost savings and framework adoption likelihood among Kubernetes-experienced respondents. Improved cost awareness (94.4%) leads expected benefits, while 100% of respondents indicate positive adoption likelihood (72.2% very likely, 27.8% likely).

Benefit/Likelihood	Count	Percentage
Expected Benefits Beyond Cost Savings		
Improved awareness and visibility of costs	17	94.4%
Improved resource utilization efficiency	15	83.3%
Real-time observability over cloud costs and sources	12	66.7%
Real-time cost predictions	11	61.1%
Framework Adoption Likelihood		
Very likely	13	72.2%
Likely	5	27.8%
Neutral	0	0.0%
Unlikely	0	0.0%
Very unlikely	0	0.0%