

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Distributed FinOps: Allocate Infrastructure Costs Across Teams for Centralized Services

João Pedro Carvalho Moreira



Master in Informatics and Computation Engineering

Supervisor: André Restivo

Co-Supervisor: João Pedro Dias

July 27, 2025

Distributed FinOps: Allocate Infrastructure Costs Across Teams for Centralized Services

João Pedro Carvalho Moreira

Master in Informatics and Computation Engineering

Approved in oral examination by the committee:

President: Prof. Nuno Macedo

Examiner: Prof. Júlio Duarte

Supervisor: Prof. André Restivo

July 27, 2025

Abstract

Organizations increasingly rely on shared cloud infrastructure to enhance scalability and resource efficiency. While this approach optimizes operational performance, it introduces significant challenges in the allocation of infrastructure costs across internal teams. Accurate cost allocation is essential not only for financial transparency but also for promoting fairness, accountability, and cost-conscious engineering.

Existing solutions that implement cost allocation based on telemetry data typically rely on a single usage metric, which poses a critical limitation. In most technologies, resource usage is inherently heterogeneous. Different dimensions, such as compute, network, and storage, each impose different cost impacts.

Additionally, current solutions fail to recognize producer-consumer relationships within organizations. This oversight often results in provider teams bearing costs incurred by consumer teams, discouraging the reuse and sharing of services.

To address these limitations, this thesis proposes a cost allocation framework that supports multi-metric cost modeling with weighted contributions, enabling accurate representation of the cost impact of diverse resource types. Furthermore, it introduces a rule-based discount mechanism to support inter-team cost redistribution, reflecting service dependencies across the organization. The framework is technology-agnostic, meaning it can be applied to any system by adapting a configuration file that defines the dimensions, metrics, weights, and discount rules.

The framework was evaluated using Apache Kafka as the target technology. A simulated scenario with production-level metrics and inter-team relationships was used to test allocations, alongside a comparative analysis with Conductor Chargeback Interceptor, covering isolated validation scenarios. Fairness was the central evaluation criterion, defined through three core principles: Open Data, Good Governance Practices, and Accountable Consumption. Open Data emphasizes rewarding teams that make their data widely available and reused across the organization. Good Governance Practices recognize teams that maintain high operational standards. Accountable Consumption ensures each team is financially responsible for its own resource usage.

The results confirm that the proposed framework enables fair cost allocation in shared infrastructure environments. Through a multi-dimensional, weighted model, it effectively differentiates between resource types and their relative cost impacts. The rule-based discount mechanism further enhances fairness by rewarding positive behaviors, shifting costs from producers to consumers, and incentivizing high-value data contributors.

Resumo

As organizações recorrem cada vez mais a infraestruturas de nuvem partilhadas para otimizar a escalabilidade e a eficiência na utilização dos recursos. Embora esta abordagem optimize o desempenho operacional, introduz desafios significativos na atribuição de custos de infraestrutura entre equipas. Uma alocação precisa de custos é essencial não apenas para garantir transparência financeira, mas também para promover equidade, responsabilização e uma engenharia mais consciente.

As soluções existentes que implementam alocação de custos com base em dados de telemetria dependem, geralmente, em uma única métrica de utilização, o que representa uma limitação crítica. Na maioria das tecnologias, a utilização de recursos é inerentemente heterogénea. Diferentes dimensões, como computação, rede e armazenamento, impõem impactos distintos nos custos.

Além disso, as soluções atuais não reconhecem as relações produtor-consumidor dentro das organizações. Esta omissão resulta frequentemente em que equipas fornecedoras suportem os custos gerados por equipas consumidoras, desencorajando a reutilização e partilha de serviços.

Para colmatar estas limitações, esta dissertação propõe uma plataforma de alocação de custos que suporta modelação multi-métrica com contribuições pesadas, permitindo representar o impacto de diferentes tipos de recursos no custo total. Além disso, introduz um mecanismo de descontos com base em regras, que permite a redistribuição de custos entre equipas, refletindo as dependências de serviços existentes na organização. A plataforma é agnóstica à tecnologia, podendo ser aplicada a qualquer sistema através da adaptação de uma configuração que define dimensões, métricas, pesos e regras de desconto.

A plataforma foi avaliada tendo como tecnologia-alvo o *Apache Kafka*. Foi criado um cenário simulado com métricas e relações entre equipas de um ambiente de produção para testar a alocação, complementado com uma análise comparativa com o *Conduktor Chargeback Interceptor*, abrangendo cenários isolados de validação. A justiça foi o critério central de avaliação, definida com base em três princípios fundamentais: Dados Abertos, Boas Práticas de Governança e Consumo Responsabilizado. Dados Abertos realça a importância de recompensar equipas que tornam os seus dados amplamente disponíveis e reutilizáveis na organização. Boas Práticas de Governança reconhecem equipas que mantêm elevados padrões operacionais. Consumo Responsabilizado assegura que cada equipa é financeiramente responsável pelos recursos que consome.

Os resultados confirmam que a plataforma proposta permite uma alocação justa de custos em ambientes de infraestrutura partilhada. Através de um modelo multidimensional e pesado, diferencia eficazmente os tipos de recursos e os seus impactos relativos nos custos. O mecanismo de desconto baseado em regras reforça a justiça ao recompensar comportamentos positivos, redistribuindo custos de produtores para consumidores e recompensando contribuidores de dados altamente consumidos.

Acknowledgements

This dissertation marks the end of my academic journey, a path filled with challenges, growth, and unforgettable moments. Reaching this milestone would not have been possible without the support of the incredible people in my life.

First and foremost, I would like to thank my mother, Lurdes Carvalho, and my father, João Paulo Moreira, for their constant support, encouragement, and belief in me throughout all these years. Their guidance and sacrifices laid the foundation for everything I've achieved.

I am also grateful to my sister, Joana Moreira, for the support and all fun moments we shared that always cheered me up and reminded me to enjoy the lighter side of life, even during stressful times.

To my stepmother, Silvia Pereira, thank you for your kindness and support over the years.

To my girlfriend, Bruna Marques, thank you for your love, patience, and understanding. Your support during the most intense moments of this journey has been invaluable, and I'm deeply grateful to have you by my side.

A special thanks goes to my two close friends, Lia Vieira and Hugo Gomes, who shared this entire academic journey with me. From the first day of classes to the final deadlines, your companionship and all shared moments made the experience all the more meaningful.

I would also like to sincerely thank my supervisors, João Pedro Dias and André Restivo, for their guidance and insights. Their mentorship has greatly shaped both this work and my personal development.

Finally, to all my family and friends, thank you. Whether through kind words, thoughtful gestures, or simply being there, your support has made a lasting impact.

João Moreira

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. Adopted in 2015 as part of the 2030 Agenda for Sustainable Development, the SDGs consist of 17 interconnected goals aimed at addressing the world's most pressing challenges, including poverty, inequality, climate change, environmental degradation, peace, and justice.

This dissertation aligns with several SDGs through its focus on improving cloud infrastructure efficiency, promoting responsible digital resource consumption, and enabling more transparent and accountable engineering practices. In particular, it contributes to:

- **SDG 9 (Industry, Innovation and Infrastructure)** by fostering more efficient and sustainable use of shared cloud environments.
- **SDG 12 (Responsible Consumption and Production)** by encouraging cost-conscious behavior and minimizing waste in cloud resource usage.
- **SDG 13 (Climate Action)** by supporting digital sustainability efforts through improved visibility into resource usage per team, which enables better planning and utilization of resources, indirectly contributing to carbon reduction.

SDG	Target	Contribution	Performance Indicators and Metrics
9	9.4	Enhancing cloud infrastructure governance through accurate cost allocation promotes more efficient and sustainable use of shared digital resources.	Reduction in unallocated cloud costs; efficiency gains in infrastructure usage; decreased over-provisioning.
12	12.6	Enables organizations to transparently report cloud resource consumption and sustainability impacts by tying costs to specific teams and services.	Number of teams with visibility into their cloud usage; adoption of sustainability reporting practices tied to cloud operations.
	12.2	Supports responsible consumption of cloud resources by reducing waste and encouraging cost-conscious behavior.	Decrease in idle or underutilized resources; optimization of memory, CPU, and storage allocation.
13	13.3	Facilitates climate-conscious engineering by linking cost and usage data to carbon footprint awareness and optimization.	Estimated CO_2 reduction through better resource efficiency; adoption of green computing practices.

Contents

1	Introduction	1
1.1	Context	1
1.2	Motivation	1
1.3	Problem definition	2
1.4	Goals	3
1.5	Research Questions	3
2	Background	5
2.1	FinOps	5
2.1.1	FinOps Principles	5
2.1.2	FinOps Lifecycle	6
2.2	Apache Kafka Overview	7
3	State of the art	9
3.1	Literature Review	9
3.1.1	Methodology	9
3.1.2	Digital Databases	9
3.1.3	Search Strings	10
3.1.4	Primary Results	10
3.1.5	Results Filtering	11
3.1.6	Main Results	12
3.2	Grey Literature	13
3.2.1	Results	13
3.3	Results Analysis	17
3.3.1	Parameters	17
3.3.2	Overview	18
3.4	Conclusion	19
4	Problem Statement	21
4.1	Problem	21
4.2	Desiderata	22
4.3	Hypothesis	22
4.4	Research Questions	23
4.5	Proposed Solution	24
5	Solution	25
5.1	Core Principles	25
5.1.1	Definition of Logical Units	26

5.2	Formal Cost Allocation Model	26
5.2.1	Base Cost Allocation	26
5.2.2	Discount Mechanism	30
5.3	Implementation	33
5.3.1	Prometheus	33
5.3.2	Architecture High Level Overview	34
5.3.3	Configuration File Specification	36
5.3.4	Cost Allocation Logic	37
6	Kafka Case Study	43
6.1	Multi-Tenant Usage	43
6.2	Metric Collection	44
6.3	Deployment Context	44
7	Evaluation and Validation	45
7.1	Definition of Fairness	45
7.2	Simulator	46
7.3	Methodology for Cost Extraction	47
7.4	Real World Scenario	47
7.4.1	Simulator Setup	47
7.4.2	Dataset Description	48
7.4.3	Cost Allocation Configuration	53
7.4.4	Experimental Setup	57
7.4.5	Results	58
7.4.6	Discussion	60
7.5	Comparative Evaluation	62
7.5.1	Experimental Setup	62
7.5.2	Cost Allocation Configuration	63
7.5.3	Discounts Configuration Per Scenario	66
7.5.4	Experiments	67
7.5.5	Discussion	77
7.6	Threats to Validity	80
8	Conclusion	82
8.1	Conclusions	82
8.2	Research Questions Revised	83
8.3	Future Work	84
	References	86
A	Cost Allocation Framework Configuration (JSON)	89
B	Simulator Configuration Example (JSON)	92
C	Service Account Dependency Graph	94

List of Figures

3.1	Systematic Literature Review filtering process diagram.	11
5.1	High-level Architecture Diagram	35
7.1	Boxplot of In-Degree Distribution	50
7.2	Boxplot of Out-Degree Distribution	50
7.3	Boxplot of Message Ingress Rates	51
7.4	Boxplot of Log Sizes	51
7.5	Top 10 Service Accounts by In-Degree (Being Consumed)	52
7.6	Top 10 Service Accounts by Out-Degree (Consuming)	52
7.7	Top 10 Service Accounts by Message Rate	53
7.8	Top 10 Service Accounts by Log Size	53
7.9	Caption for the symbols used in the scenarios diagram	67
7.10	Scenario S.01 Diagram	68
7.11	Scenario S.02 Diagram	69
7.12	Scenario S.03 Diagram	70
7.13	Scenario S.04 Diagram	71
7.14	Scenario S.05 Diagram	72
7.15	Scenario S.06 Diagram	73
7.16	Scenario S.07 Diagram	74
7.17	Scenario S.08 Diagram	75
7.18	Scenario S.09 Diagram	77
C.1	Service Account Dependency Graph	94

List of Tables

3.1	Search Strings for Literature Review	10
3.2	Number of search results for each query in different databases	11
3.3	Inclusion criteria for filtering of the primary results	11
3.4	Exclusion criteria for filtering of the primary results	11
3.5	Comparison of Cost Allocation Solutions	18
5.1	Examples of metric labels and how regex patterns extract logical units.	28
7.1	Statistical Summary Scenario Metrics	50
7.2	Statistical Summary Results	59
7.3	Spearman Correlation Coefficients between Scenario Metrics and Allocation Variables	60
7.4	Summary of Discount Application per Scenario	67
7.5	Scenario S.01 Allocation Results	68
7.6	Scenario S.02 Allocation Results	69
7.7	Scenario S.03 Allocation Results	70
7.8	Scenario S.04 Allocation Results	71
7.9	Scenario S.05 Allocation Results	72
7.10	Scenario S.06 Allocation Results	73
7.11	Scenario S.07 Allocation Results	74
7.12	Scenario S.08 Allocation Results	76
7.13	Scenario S.09 Allocation Results	77

Listings

7.1	JSON configuration Dimensions	55
7.2	JSON configuration High Topic Reader Discount	56
7.3	JSON configuration High Ratio Bytes Out/In Discount	57
7.4	JSON configuration Dimensions for Multi-AZ Deployment	64
7.5	JSON configuration Low Consumer Lag	65
7.6	JSON configuration Schema Compatability	66
A.1	Cost Allocation Configuration (JSON)	89
B.1	Simulator Configuration Example(JSON)	92

List of Algorithms

1	Base Allocation	38
2	Discount Eligibility and Extraction	40
3	Global Redistribution (<i>penalized_label</i> = <i>_all</i>)	40
4	Proportional Strategy	41
5	Equal Strategy	41

List of Acronyms

IDP	Internal Developer Platform
SA	Service Account
SLR	Systematic Literature Review
AWS	Amazon Web Servers
CSP	Cloud Service Provider
GCP	Google Cloud Platform
AZ	Availability Zone
EC2	Elastic Compute Cloud
EBS	Elastic Block Store
EKS	Elastic Kubernetes Service

Chapter 1

Introduction

1.1 Context

In today's technological landscape, modern organizations increasingly rely on cloud infrastructure to support their operations. This infrastructure is often shared among multiple teams, which allows for greater scalability and resource efficiency. In response to growing complexity, many have introduced shared environments and Internal Development platforms (IDPs).

An IDP "consists of many different techs and tools, glued together in a way that lowers cognitive load on developers without abstracting away context and underlying technologies"[44]. Typically, these platforms are maintained by a dedicated team but are used by multiple teams across the organization. While such sharing improves resource utilization and developer efficiency, it also introduces new challenges in cost management, particularly in attributing cloud costs back to the responsible teams or business units. In those scenarios where one team provides services consumed by another, accurately attributing costs to the teams that utilize these shared resources becomes especially critical to ensure fair cost distribution and prevent any single team from disproportionately bearing the financial burden.

In this multi-team environment where services coexist on shared infrastructure, the cloud provider's bill is aggregated, and splitting it accurately by team is non-trivial. The need to attribute costs to the right owner (team, product, or department) has thus become a crucial yet complex aspect of cloud governance.

1.2 Motivation

Accurate cloud cost attribution is not merely a technical optimization but an enabler of fairness, accountability, and operational efficiency in modern organizations. As organizations expand their cloud infrastructure and rely more heavily on shared platforms, it becomes increasingly important to ensure that each team is accurately charged for the resources it consumes. Without reliable attribution mechanisms, cloud costs are often centralized into a single shared budget, obscuring which teams are responsible for specific portions of spend. This lack of visibility creates fairness

issues; some teams may be overcharged, others undercharged, leading to disputes and poor budget predictability.

When teams cannot see the costs they generate, there is no incentive to manage usage or reduce inefficiencies. Conversely, when costs are transparently attributed, teams gain ownership over their cloud spend and are more likely to take action. This sense of accountability fosters a culture of cost-conscious engineering, where developers and product owners actively consider the financial implications of their architectural and scaling decisions.

A core FinOps principle is that “everyone takes ownership of their cloud usage” [20] – in practice, this means each team should understand and be accountable for the costs they drive. By associating real infrastructure costs with services and teams, organizations empower individuals to act on data and optimize spend where it matters most.

In addition to driving responsible engineering behavior, accurate attribution significantly improves forecasting and budgeting. Financial planning becomes more precise when historical usage and cost patterns are tied to specific teams or services. Teams can better estimate future spend and detect anomalies early. Without this granularity, organizations are left to make strategic decisions based on incomplete or aggregated data, leading to reactive rather than proactive cost management. Moreover, as shared infrastructure grows in size and complexity, so too does the volume of unallocated costs. Without proper redistribution, these costs become harder to predict and control, increasing the risk of budget overruns.

Cost attribution also intersects with sustainability goals. Many organizations are actively pursuing carbon reduction strategies alongside cost optimization efforts. Poor cost visibility can lead to over-provisioning or underutilized resources, not only inflating cloud bills but also resulting in unnecessary energy consumption. By making resource usage visible at the team level, organizations can identify and eliminate waste, achieving both financial and environmental efficiency.

These motivations are not abstract. They are evident in the case of Kuehne+Nagel [29], a global logistics company, which serves as the industry partner for this dissertation. Like many large organizations, they face challenges in attributing the cost of shared services, such as Kafka clusters or internal developer tooling, back to the teams that consume them.

In summary, the motivation for this research lies in enabling accountable cloud operations. Accurate cost attribution creates a virtuous cycle: teams are accountable for their cost, reduce waste, improve forecasting, and contribute to more financially and environmentally responsible engineering. In a cloud-centric world, where every service relies on shared infrastructure, getting attribution right is not optional; it is essential to aligning technology with business value.

1.3 Problem definition

There are three standard methods for distributing shared costs [21]. The first approach is *even splitting*, where the total amount is divided equally across teams. While this method is straightforward and often used in smaller organizations due to its simplicity, it can become unfair or even inappropriate in larger companies, disproportionately affecting the budgets of certain units.

The second method is *fixed proportional allocation*, which involves using a set percentage to attribute shared costs. These ratios are typically determined by analyzing past spending to determine a fair cost distribution. Although this strategy is generally more equitable than even splitting, its effectiveness depends on how frequently the ratios are updated to reflect changing usage patterns.

The third and final approach is *variable proportional allocation*, which allocates costs based on each team's actual resource consumption percentage. This method adjusts automatically, making it the most dynamic and responsive strategy for cost allocation.

All cost allocation solutions empower at least one of these strategies. Solutions that implement the *variable proportional allocation* strategy often rely on a single usage metric. This simplification fails to account for the diverse cost profiles of modern cloud workloads, where resource types like memory, storage, and network can vary widely in their impact on cost. For example, a data processing service may use minimal CPU but generate high storage and network costs, an imbalance not captured by CPU-based allocation alone.

In addition, associating resource usage with logical organizational entities (e.g., teams, departments, or projects) is non-trivial. Most systems depend on metadata such as tags or labels.

A further complication arises when services provided by one team are consumed by others. Without mechanisms to redistribute costs based on these internal dependencies, provider teams may be unfairly charged for shared infrastructure, disincentivizing service reuse and collaboration.

The limitations of these strategies and their implementation in current industry solutions are analyzed in detail in Chapter 3.

1.4 Goals

The primary goal of this dissertation is to improve the accuracy and fairness of cost allocation in shared infrastructure environments.

To achieve this, the dissertation proposes the design and implementation of a FinOps-based cost allocation framework. The framework builds upon the variable proportional allocation strategy by incorporating multiple usage metrics, accounting for the differentiated cost impact of resource types, and supporting cost redistribution between provider and consumer teams based on inter-service dependencies.

Once the framework is implemented, its effectiveness is evaluated through real-world scenarios. The evaluation focuses on the improvements it brings to the cost allocation process in terms of fairness..

1.5 Research Questions

This dissertation addresses the challenge of fair and accurate cloud cost attribution in organizations that operate shared infrastructure. It proposes a cost allocation framework that builds on the principles of FinOps and seeks to go beyond traditional single-metric allocation strategies.

The research is guided by the central hypothesis that:

“In organizations that operate shared infrastructure, a framework that allocates costs based on multiple metrics, incorporates the differentiated cost impact of resource types, and supports inter-team cost redistribution will improve fairness.”

To validate this hypothesis, the study investigates the following research questions:

- **RQ1:** How can telemetry data be used to allocate infrastructure costs in shared environments?
- **RQ2:** How can resource usage be accurately mapped to organizational entities (e.g., teams, departments, or projects) using available metadata?
- **RQ3:** What is the impact of considering multiple metrics with differentiated cost impact on the fairness of cost allocation?
- **RQ4:** What approaches can be used to determine the relative contribution or importance of each metric in the allocation model?
- **RQ5:** How can the cost allocation be extended to incorporate inter-team service relationships and incentivize efficient, collaborative infrastructure usage?
- **RQ6:** What validation techniques and evaluation metrics can be applied to assess the fairness of the proposed cost allocation framework?

These questions explore how telemetry data can be leveraged, how resource usage can be mapped to organizational entities, how different metrics influence fairness, and how cost redistribution mechanisms can be applied to reflect inter-team service relationships.

A full discussion of the hypothesis and research questions is presented in Chapter 4.

Chapter 2

Background

This chapter introduces concepts that will be useful to better comprehend this dissertation. Section 2.1 introduces the FinOps framework, outlining its core principles and lifecycle. Section 2.2 provides an overview of Apache Kafka, including its architectural and main concepts.

2.1 FinOps

The adoption of cloud computing has introduced a paradigm shift in how organizations manage their technology and financial operations. The elasticity, scalability, and on-demand nature of cloud services bring significant advantages, but they also introduce a new level of complexity in cost management. Traditional IT financial practices, designed for predictable, fixed-cost environments, struggle to address the dynamic and granular nature of cloud billing. This disconnect has led to the emergence of a new operational framework: FinOps.

FinOps is defined by the FinOps Foundation as "an operational framework and cultural practice which maximizes the business value of cloud and technology, enables timely data-driven decision making, and creates financial accountability through collaboration between engineering, finance, and business teams." [17]. It blends disciplines from finance, engineering, and operations to deliver a unified approach for managing cloud costs.

2.1.1 FinOps Principles

The FinOps framework is grounded in six key principles [20] that shape both the cultural and operational practices necessary for managing cloud costs effectively.

The first principle emphasizes that "teams need to collaborate." In a cloud environment where spending occurs continuously and dynamically, finance, engineering, product, and business teams must work together in near real time. This cross-functional collaboration ensures that cost-related decisions are aligned with business goals and that teams can respond quickly to inefficiencies or changing priorities.

The second principle asserts that "business value drives technology decisions." Instead of focusing solely on aggregate spend, organizations are encouraged to evaluate cloud usage in terms

of unit economics and value-based metrics. Trade-offs between cost, speed, and quality are made consciously, with cloud viewed as a strategic enabler rather than just an operational expense.

The third principle holds that “everyone takes ownership of their technology usage.” Engineers and product teams are expected to manage and take responsibility for the cost impact of their architectural and operational decisions. To support this, cost data must be timely, granular, and widely accessible throughout the organization. Transparency in spending encourages more efficient behavior and enables rapid feedback loops.

The fourth principle states that “FinOps data should be accessible, timely, and accurate.” Real-time visibility into cloud spend is critical for driving efficient behavior and supporting fast feedback loops. Cost and usage data must be processed and shared as soon as it becomes available, enabling stakeholders at all levels of the organization to act on it. This principle supports the development of accurate forecasting, variance analysis, benchmarking, and performance tracking.

The fifth principle emphasizes that “FinOps should be enabled centrally.” While cost ownership is distributed across teams, a centralized FinOps should be central to provide governance, tooling, and best practices. This central team facilitates standardization, without removing decision-making power from individual teams.

The sixth and final principle encourages organizations to “take advantage of the variable cost model of the cloud.” Unlike fixed-cost infrastructure, the cloud offers flexibility that can be leveraged for competitive advantage. Agile planning, real-time forecasting, and proactive system design allow organizations to align spending with actual usage. Rather than fearing variability, FinOps encourages embracing it as a tool for increasing efficiency and delivering more value.

2.1.2 FinOps Lifecycle

To operationalize these principles, the FinOps framework is structured around a continuous three-phase lifecycle: Inform, Optimize, and Operate [18].

In the Inform phase, the goal is to focus on gaining visibility into cloud spend. This involves collecting, organizing, and allocating cloud cost and usage data to the appropriate business units, teams, or products. The objective is to empower stakeholders with timely, accurate, and actionable information to support forecasting, budgeting, and performance benchmarking.

The Optimize phase builds on this foundation by identifying opportunities to increase cloud efficiency. This includes both rate optimization, through mechanisms like Reserved Instances or Savings Plans, and usage optimization, such as rightsizing underutilized resources. Optimization is a collaborative effort, relying on insights from the Inform phase to target the most impactful changes.

Finally, in the Operate phase, organizations embed FinOps practices into their day-to-day workflows. This includes implementing governance structures, defining roles and responsibilities, automating processes, and fostering a culture of continuous improvement. The cycle then repeats, with each phase feeding into the next to create a dynamic, iterative model of financial operations.

The problem approached in this dissertation, cost allocation, is situated within the Inform phase, where visibility and accurate attribution of cloud spend form the foundation for all subsequent FinOps activities.

2.2 Apache Kafka Overview

Apache Kafka [4] is a high-throughput, low-latency distributed event streaming platform designed to handle real-time data feeds at scale. Originating at LinkedIn and later open-sourced through the Apache Software Foundation[43], Kafka has become foundational in modern event-driven architectures due to its robust scalability, fault tolerance, and persistence guarantees. Kafka operates fundamentally as a distributed publish-subscribe (pub/sub) messaging system, structured to decouple data producers and consumers while supporting high-volume, real-time data processing.

At its essence, Kafka revolves around the concept of events that signify a particular occurrence at a specific moment in time. An event in Kafka typically includes a key, a value, a timestamp, and optional metadata headers. Events are serialized in formats such as Apache Avro[3], which offers compact binary serialization with schema evolution support.

Kafka enforces structure through schemas, which act as contracts that validate data integrity and compatibility between producers and consumers. These schemas are not natively enforced by Kafka, necessitating auxiliary tools such as Schema Registry, which store and manage these contracts, thereby facilitating data governance and consistency across services.

The primary organizational unit within Kafka is the topic, which functions like a category or feed to which records are published. Topics are divided into partitions, the fundamental units of parallelism and scalability in Kafka. Each partition is an immutable, ordered sequence of records, and Kafka appends new records to the end. Message ordering is guaranteed only within a single partition, not across the entire topic.

Kafka clients are either producers or consumers. Producers are responsible for publishing records to topics, typically distributed across partitions either randomly or via a deterministic strategy based on a message key. Consumers, in contrast, read records from topics and can be organized into consumer groups, where each member processes data from a subset of partitions. Kafka maintains the position of each consumer in a partition via an offset, an incrementing identifier that uniquely marks each record's position. This facilitates replayability, allowing consumers to reprocess historical data deterministically.

Kafka runs on a cluster of servers called brokers. Each broker handles client requests for data ingest (producers) and retrieval (consumers), stores data persistently on disk, and maintains partition logs. Kafka clusters can scale horizontally by simply adding more brokers, each of which may manage multiple partitions.

Kafka ensures reliability through a replication model where each partition has a configurable number of replicas. One broker acts as the leader for a given partition and handles all reads and writes, while other brokers maintain follower replicas for redundancy. Kafka employs a replication

protocol that ensures data is not acknowledged until it is committed to all in-sync replicas, providing durability guarantees and enabling fault tolerance: in the event of broker failure, a follower can be promoted to leader with no data loss, assuming proper configuration.

Coordination and consensus within Kafka clusters have historically relied on Apache ZooKeeper [5], which manages metadata such as broker availability, topic configurations, and leader election. ZooKeeper acts as a distributed configuration and synchronization service, ensuring a consistent view of the cluster state. However, recent Kafka versions have introduced "KRaft" mode, a built-in consensus mechanism that replaces ZooKeeper and manages metadata natively within Kafka itself.

Beyond the core Apache Kafka project, several commercial and open-source Kafka distributions have emerged, offering enhancements in usability, performance, and ecosystem integration. Confluent Platform [13], developed by the original creators of Kafka, extends Kafka with enterprise-grade features such as schema validation, stream processing with ksqlDB, advanced security, and fully managed cloud services. Redpanda [35] is a Kafka-compatible streaming platform offering lower latency and simplified operations. Conduktor [11] focuses on improving developer experience with graphical interfaces for managing Kafka clusters, inspecting topics, debugging consumer groups, and monitoring broker activity. These distributions provide additional tooling and optimizations that streamline Kafka adoption in production environments, especially for organizations requiring enhanced operability and reduced infrastructure complexity.

Chapter 3

State of the art

This chapter explores state-of-the-art solutions, both academic and industrial, for addressing the problem of cost allocation in shared infrastructure. First, a systematic literature review was conducted, as described in Section 3.1. In Section 3.2, a grey literature review was carried out to identify available industry solutions. The results obtained from these approaches are analyzed and compared in Section 3.3. Finally, Section 3.4 provides a conclusion summarizing the state of the art as a whole.

3.1 Literature Review

A Systematic Literature Review (SLR) was conducted to identify relevant research. An SLR is a structured and methodical approach designed to systematically collect, identify and analyze existing research. This approach ensures a comprehensive and unbiased overview of the academic landscape by adhering to clearly defined steps and inclusion and exclusion criteria. The following subsections outline the methodology and steps undertaken to conduct the SLR.

3.1.1 Methodology

The SLR process begins by defining search queries tailored to the research objectives. These queries are then executed in the selected databases to retrieve relevant studies. The resulting publications are subsequently filtered and analyzed.

3.1.2 Digital Databases

For the SLR, four digital databases were selected: IEEE Xplore[28], ACM Digital Library[1], Scopus[38] and Science Direct[37]. These databases were chosen for their extensive coverage of academic publications and their relevance to the fields of technology.

3.1.3 Search Strings

To conduct a literature search in the mentioned databases, three search strings were created. Initially, these search strings underwent significant modifications based on the relevance assessment of the initial results. The final search strings are shown in table 3.1.

Table 3.1: Search Strings for Literature Review

QS1	"cloud" AND ("ShowBack" OR "ChargeBack")
QS2	"cloud" AND ("cost allocation" OR "cost tagging" OR "cost categorization")
QS3	"cloud" AND ("multitenancy" OR "multi-tenant" OR "shared resources") AND ("resource usage" OR "resource utilization") AND ("monitoring" OR "telemetry")

3.1.3.1 Search Query QS1

The search string QS1 is designed to capture literature related to showback and chargeback solutions. Chargeback is a cloud cost allocation method in which cloud costs are billed to business units based on their actual usage. Showback, on the other hand, tracks and reports the costs incurred by different business units or departments without directly billing them. Both approaches are significant to this dissertation as they address cost allocation in a shared environment.

3.1.3.2 Search Query QS2

This search string is designed to identify literature on allocating costs in cloud environments. By incorporating terms like "cost allocation," "cost tagging," and "cost categorization", it focuses on studies that discuss methods for distributing cloud costs across logical units, such as teams, projects, and departments.

3.1.3.3 Search Query QS3

This search string targets research related to resource usage tracking and monitoring in multi-tenant or shared cloud infrastructures. By including terms like "multitenancy", "multi-tenant" and "shared resources", it emphasizes studies on shared environments. Additionally, "resource usage" and "resource utilization" highlight the focus on measuring or analyzing consumption patterns. The inclusion of "monitoring" and "telemetry" ensures that the search captures technical implementations or methodologies for extracting usage data.

3.1.4 Primary Results

The table 3.2 presents the number of results retrieved for each query string (QS1, QS2, QS3) from the chosen digital databases. These results represent the number of initial findings obtained without applying any filters or exclusions. The total number of primary results obtained is 225.

Table 3.2: Number of search results for each query in different databases

Query	IEEE	ACM	Scopus	Science Direct
QS1	8	13	18	2
QS2	41	6	24	4
QS3	89	3	17	0
Total	138	22	59	6

3.1.5 Results Filtering

After retrieving all the results from the digital databases, a filtering process was applied to identify the most relevant studies. To support this process, a set of inclusion and exclusion criteria was defined, as presented in Tables 3.3 and 3.4.

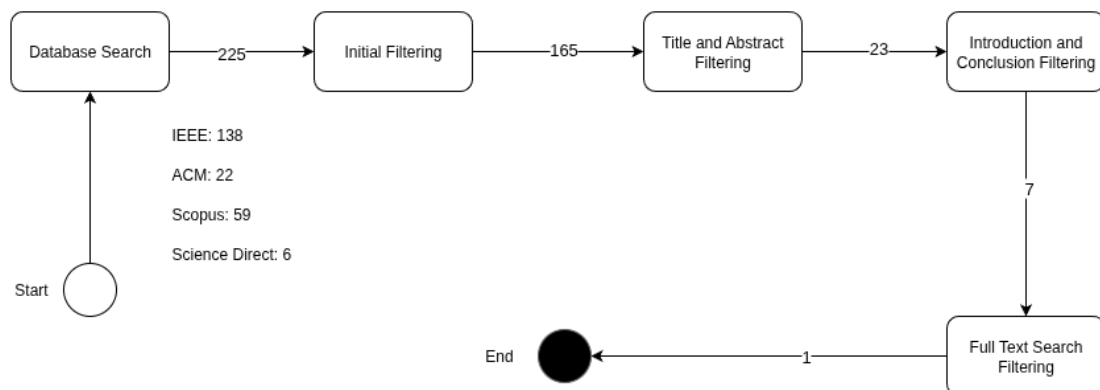


Figure 3.1: Systematic Literature Review filtering process diagram.

Table 3.3: Inclusion criteria for filtering of the primary results

ID	Description
IC1	The work addresses cost allocation in cloud environments.
IC2	The work examines resource usage monitoring or telemetry in shared cloud environments.
IC3	The work addresses chargeback or showback models in cloud infrastructure.

Table 3.4: Exclusion criteria for filtering of the primary results

ID	Description
EC1	The work is duplicated.
EC2	The work is not in English.
EC3	The work addresses the problem solely from the provider's or data center's perspective.

The exclusion criteria eliminate works that are duplicated (**EC1**) or not accessible in English (**EC2**). Additionally, works that address the problem solely from the perspective of providers or data centers are excluded (**EC3**), as the research focuses on cost allocation mechanisms from the viewpoint of teams using shared cloud infrastructure.

The inclusion criteria ensure that the selected research is directly relevant to this study. **IC1** prioritizes works on cost allocation in cloud environments, as this forms the core of the research. Studies examining resource usage monitoring or telemetry in shared cloud environments (**IC2**) are included because these aspects are essential for understanding consumption patterns and enabling accurate cost allocation. Furthermore, research that addresses chargeback or showback models in cloud infrastructure (**IC3**) is included.

The following process was implemented to filter the primary results: First, an **initial filtering** step was conducted to remove duplicate studies and those not written in English, as specified by **EC1** and **EC2**. Next, a **metadata filtering** phase was applied, where the metadata of each paper was analyzed to identify studies relevant to the field based on the inclusion criteria (**IC1**, **IC2**, and **IC3**) and exclusion criterion **EC3**. Following this, an **introduction and conclusion filtering** step was performed, applying the same relevance checks but focusing on the introduction and conclusion sections of each study. Finally, a **full-text filtering** step was undertaken, expanding the analysis to the complete content of the selected documents.

3.1.6 Main Results

- **Baars et. al** [7] explores the complexities of implementing chargeback mechanisms in cloud environments. The paper identifies and validates eight key factors that influence the success of cloud-specific chargeback systems: Accuracy, Cost of Costing, Transparency and Understandability, Controllability, Fairness, Predictability, Measurability and Comparability. These factors are derived through a systematic literature review and refined through empirical research, including expert interviews and workshops. For the author the interpretation of the key-factors is the following:
 - *Accuracy* refers to the degree to which the charges allocated to an organizational unit accurately reflect the actual costs incurred in delivering services to that unit.
 - *Cost of Costing* measures the expenses associated with designing, implementing, and operating a chargeback system. It includes costs related to metering, data collection, and system maintenance. The challenge is to balance these costs against the benefits.
 - *Transparency and Understandability* reflects the ability of stakeholders to comprehend how charges are calculated and allocated,
 - *Controllability* describes the extent to which users can influence their IT bills by modifying their consumption behavior. A chargeback model with high controllability empowers users to make cost-conscious decisions, aligning usage patterns with budgetary goals.

- *Fairness* is the degree to which stakeholders perceive the chargeback model as equitable. It includes both allocative fairness (ensuring costs are distributed accurately based on consumption - accuracy) and perceived fairness (how fair the model feels to stakeholders). Even if the system is technically accurate, it can be deemed unfair if it disproportionately impacts certain units or does not align with organizational objectives.
- *Predictability* concerns the ability of stakeholders to foresee their IT costs based on known consumption patterns.
- *Accountability* ensures that users can verify the correctness of their bills. A chargeback system with high accountability provides users with the tools and data needed to trace charges back to their source, thereby minimizing disputes and fostering trust in the system.
- *Measurability* refers to the ease and accuracy of tracking resource usage and attributing it to specific users or units. High measurability reduces ambiguity in billing, ensures accurate allocation, and builds confidence in the fairness of the chargeback model.
- *Comparability* allows users to benchmark the prices of internal services against equivalent offerings in the market.

The authors categorize these factors into two impact dimensions: effectiveness, which measures how well the system achieves cost awareness and acceptability, which refers to the acceptance of stakeholders to the chargeback process. The author considers that Predictability, Transparency Understandability, and Accountability positively impact the acceptability dimension, while Comparability positively affects the effectiveness dimension. Additionally, the factors of Measurability, Accuracy, and Controllability positively impact both dimensions. Finally, the Cost of Costing negatively affects the effectiveness dimension.

3.2 Grey Literature

In addition to the SLR, multiple manual Google searches were conducted to identify relevant projects or solutions available in the market. This search revealed the following products.

3.2.1 Results

- **CloudZero** [10] is a cloud cost optimization platform designed to help organizations save money and promote accountability. It supports cost tracking from multiple sources, including AWS, Snowflake, Azure, GCP, Datadog, New Relic, MongoDB, and Databricks. Additionally, it provides the ability to create custom cost source adapters. The most valuable feature of CloudZero for this thesis is its CostFormation(custom cost allocation rules), which uses a YAML-based configuration. This feature allows users to define Custom Dimensions, consisting of rules and conditions to assign cloud charges to various elements.

For instance, a Dimension could represent the environment in which a resource is deployed, such as Production or Development. By specifying rules, for example, in a scenario where accounts A and B are for a Production environment, assigning charges to the Production environment when the account equals A or B, users can attribute costs to specific elements of the Dimension. Alongside Dimension definitions, CostFormation enables users to create rules for splitting shared costs. These rules allow costs to be allocated proportionally or evenly among different elements. An alternative method for shared cost allocation involves using telemetry streams. CloudZero supports ingesting custom usage or business metrics to drive cost allocation. It provides a Telemetry API and CSV import to send any external metrics to CloudZero. These metrics can then be used in Telemetry Allocation Dimensions to allocate costs based on external usage data. There isn't evidence of CloudZero allowing a shared cost to be split by a weighted combination of multiple different metrics simultaneously. Instead, you would choose one metric as the driver per allocation rule.

- **Conduktor Chargeback Interceptor** [8] - Conduktor [11] is a platform designed to streamline the management and observability of Apache Kafka environments. It provides a suite of tools for developers, operators, and platform teams to interact with Kafka clusters more effectively, offering capabilities for data governance, schema management, auditing, security, and monitoring.

Within Conduktor's architecture, an interceptor is a lightweight plugin component that intercepts and inspects Kafka traffic in real time. Interceptors are deployed within the Conduktor Gateway, a network proxy. This proxy-based design allows interceptors to observe, transform, or enrich Kafka protocol communications without requiring changes to the Kafka clients or brokers themselves. Interceptors can be used for logging, metrics collection, access control, security enforcement, and, in this context, cost attribution.

The Chargeback interceptor records metrics related to data production and consumption, specifically aggregating bytes produced and consumed by service accounts. These metrics are then transmitted to a dedicated Kafka topic, which serves as the foundation for the chargeback feature. Using the collected metrics, the Chargeback system calculates the cost per Service Account based on a user-defined price per GB of data produced and consumed. The Chargeback UI provides an interface for visualizing cost and usage data. Users can view metrics in either graphical or tabular form, from multiple analytical perspectives:

- **Overall Cost:** Displays the total cost by summing both ingress (produced) and egress (consumed) traffic.
- **Ingress:** Shows the volume of data produced by each Service Account, expressed in bytes, percentages, or cost.
- **Egress:** Shows the volume of data consumed by each Service Account, similarly visualized.

- **Harness** [26] provides a cost management module as part of its broader platform. The tool integrates with major cloud providers (AWS, GCP, Azure) and Kubernetes environments to provide visibility into cloud spending. Its primary mechanism for cost attribution is through *Cost Categories*, which serve as a way to assign cloud costs to logical units, such as teams or departments, across multiple data sources.

A *Cost Category* in Harness is composed of cost buckets, which are defined using boolean rules applied to resource metadata, such as account IDs, cloud native tags, or cluster identifiers. These buckets aggregate usage data from different providers and environments to produce a unified view of cost per logical unit. For example, if a team's resources are distributed across several AWS accounts and GCP projects, all associated costs can be grouped into a single bucket labeled with that team's name.

Harness supports shared cost allocation through the use of shared buckets, allowing for the distribution of expenses related to shared infrastructure. Three allocation strategies are supported:

- **Equal Split:** Divides the cost evenly among selected buckets.
- **Proportional Split:** Allocates the shared cost based on each bucket's existing spend. For example, if team A incurs 60\$ and team B incurs 40\$, then the shared cost is split 60/40.
- **Fixed Percentage:** Allows manual assignment of exact percentages.

Harness supports ingesting cost data from external sources via a feature called *External Cost Data Ingestion*. This allows you to import costs from SaaS vendors or other services such as Mongo or Snowflake, into Harness in a standard CSV format (FOCUS). By doing so, you can include those external costs in your Perspectives and cost reports for a unified view. However, there's no evidence that Harness ingests arbitrary telemetry metrics for cost allocation purposes.

- **Doit** [16] is a tool for cloud analytics, automation, and financial management. The most relevant feature of this tool for this thesis is the *Split Shared Costs* feature. Doit introduces the concept of *Attributions*. An *Attribution* is a logical grouping of cloud resources that defines a cost category; each *Attribution* has a set of properties that map a dimension to a value. Doit offers multiple dimension types, such as the cloud provider, a country, region, a Project/Account ID, a Project/Account name, resource metadata, and resource labels (AWS cost allocation tags, Google Cloud labels, Azure tags), among others. These properties are then combined using Boolean operators. For example, if we wanted to create an Attribution for the Engineering team, and all cloud resources used by this team are either in AWS Account A or AWS Account B and are tagged with "team:engineering," an Attribution with three dimensions could be created: two Account ID dimensions (Account A and Account B) and one label dimension (tagged with team:engineering). These dimensions would be

combined using an OR operator between the Account ID dimensions and an AND operator with the label dimension.

DoiT also introduces the concept of *Attribution Groups*, which are collections of Attributions that may or may not share costs. These Attribution Groups allow splitting costs across the Attributions. This tool offers three methods to split costs: the Even Split method, which divides costs evenly between the Attributions; the Proportional Split method, which splits costs in proportion to the total value of each Attribution; and the Custom Split method, which allows users to manually define the proportions for each Attribution.

DoiT provides a feature called DataHub, which enables ingestion of third-party cost data into the DoiT console. DataHub supports importing data via API or CSV. DoiT's proportional split is based on cost share, not a separate usage metric.

- **Finout** [19] is a cloud cost management platform that unifies billing data from multiple cloud providers, offering integrations with AWS, Azure, Oracle, and GCP. For AWS, Finout uses the Cost and Usage Report (CUR) to monitor cloud expenses. It provides a centralized dashboard, the MegaBill, which enables cost monitoring and usage tracking. Usage data is currently available for AWS, GCP, and Azure.

Finout introduces Virtual Tags, a feature that enables tagging of costs based on custom rules. A rule consists of *where* clauses and a *then* clause. The *where* clauses match specific costs based on attributes like cost center (e.g., AWS, GCP), keys (typically native tags from providers), operators (e.g., One Of, Not One Of, Is, Is Not, Contains, Not Contains, Exists, or Not Exists), and one or more values. A match is determined when the key's value, in the cost data, aligns with the specified values using the chosen operator. Complex rules can be built by combining multiple *where* clauses with boolean operators (AND/OR). The *then* clause specifies the custom value to tag the matched resources.

Virtual Tags address the challenge of allocating costs to specific owners or projects. However, shared costs, which cannot be directly attributed, require additional allocation strategies. Finout provides two methods for allocating shared costs: Telemetry-Based Reallocation and Customized Cost Reallocation.

Customized Cost Reallocation allows to split the costs either by an equal split approach or a fixed percentage approach.

Telemetry-Based Reallocation uses external telemetry data to distribute shared costs. Finout supports telemetry integration through four methods:

- **S3 Bucket Telemetry Export:** Data is exported to an S3 bucket accessible by Finout, which checks the bucket daily for new CSV files.
- **Datadog Integration:** Telemetry data is retrieved from Datadog.
- **Snowflake Integration:** Telemetry data is sourced from Snowflake.

- **Prometheus Integration:** Supported exclusively for Kubernetes environments monitored by Prometheus.

Finout calculates allocation percentages daily based on the telemetry data provided by these integrations. However, from what we found, Finout’s telemetry-based allocation only allows choosing one metric at a time to split a given cost. The interface will have you select a single telemetry stream when configuring a shared cost reallocation.

- **Vantage.sh** [42] is a cloud cost management platform. Among its core features, Vantage provides mechanisms for cost allocation across organizational units, primarily through two techniques: tag-based attribution and dynamic shared cost redistribution.

Cost allocation in Vantage is implemented via *Cost Allocation Segments*, which allow users to define logical groupings (e.g., teams or services) based on metadata such as account IDs, tags, or service types. These segments form a hierarchical structure in which cloud expenses are assigned to unique paths. The system relies on user-defined rules, derived from available billing data (such as the AWS Cost and Usage Report).

To address untagged or shared resources, Vantage introduces *Dynamic Cost Allocation*. This feature enables the proportional redistribution of shared costs to defined segments using another set of existing costs or business metrics. Business metrics in this context refer to indicators that represent operational activity or service consumption. These metrics can be imported via CloudWatch, Datadog, CSV uploads, or through a direct API. Once imported, these metrics can be mapped to business units and used to determine the relative share of a shared cost. However, Vantage’s *Dynamic Allocation* uses one chosen metric at a time for a given allocation rule. We did not find any capability to combine multiple metrics.

3.3 Results Analysis

In this section, a comparison of all the solutions is made across various parameters. **Baars et al.** [7] is excluded, as it is not a solution but rather a methodology for validating a solution.

3.3.1 Parameters

- **Technology-oriented or CSP-oriented:** Is the tool primarily technology-oriented, focused on telemetry from target systems, or cloud-provider-oriented, mainly relying on cloud provider billing and metadata?
- **Logical unit correlation** How does the tool associate costs with logical entities (teams, services, projects, etc.)?
- **Supports custom telemetry** Does the tool allow the use of custom or external telemetry data to drive cost allocation?
- **Metric Source** What types of metric sources are supported for cost calculation?

- **Multi-metric support** Can the tool combine multiple metrics in the cost allocation model?
- **Metric weighting** Does the tool support assigning weights to different metrics to reflect their relative cost impact?
- **Shared cost allocation strategies** Which strategies are supported for shared costs (equal split(ES), fixed percentage(FP), telemetry-based proportional split(TBP)?
- **UI/Dashboarding** Does the tool provide a built-in interface for visualizing cost breakdowns?

Table 3.5: Comparison of Cost Allocation Solutions

Tool	Technology or CSP oriented	Logical Unit Correlation	Custom Telemetry	Metric Source	Multi-metric support	Metric Weighting	Shared Cost Strategies	UI
CloudZero	CSP-oriented	Dimensions defined by metadata and CSP tags	Yes	Custom telemetry via API or CSV	No	No	ES, FP, TBP	Yes
Conduktor	Technology-oriented	Kafka service accounts	No	Kafka metrics	No	No	TBP	Yes
Harness	CSP-oriented	Cost Categories with rule-based buckets defined via metadata	No	CSP billing	No	No	ES, FP	Yes
Doit	CSP-oriented	Attributions built from metadata and CSP tags	No	CSP billing	No	No	ES, FP	Yes
Finout	CSP-oriented	Virtual Tags based on metadata and CSP tags	Yes	CSP billing, Prometheus, Datadog, Snowflake and S3	No	No	ES, FP, TBP	Yes
Vantage.sh	CSP-oriented	Cost Allocation Segments created from metadata and CSP tags	Yes	CSP billing, Datadog, CloudWatch	No	No	TBP	Yes

3.3.2 Overview

Table 3.5 provides a comparative overview of the cost allocation tools, contrasting them across the parameters outlined in Section 3.3.1. The analysis emphasizes not only whether it is focused on the cloud provider or the technology, but also the mechanisms by which they associate costs with logical units, support for telemetry, and the flexibility of their allocation strategies.

All analyzed tools are fundamentally CSP-oriented, relying primarily on cloud provider billing data and native resource metadata as their core data sources. The exception is Conduktor Chargeback Interceptor, which is technology-oriented and derives cost attribution exclusively from Kafka protocol metrics intercepted in real-time traffic.

Logical unit correlation is achieved in different ways. CloudZero, Doit, Finout, and Vantage.sh allow users to define logical groupings (such as teams or projects) using flexible rule engines that

operate on metadata (cloud native tags, account IDs) and, in some cases, custom telemetry labels. Harness follows a similar approach using Cost Categories and buckets, where rules aggregate resource usage by tags. Conduktor instead links costs directly to Kafka Service Accounts as logical units, as extracted from network traffic metadata.

Custom telemetry support is uneven. CloudZero, Finout, and Vantage.sh accept externally ingested telemetry via APIs or CSV uploads, which can be used for cost attribution or proportional shared cost reallocation. Conduktor's custom telemetry is limited to protocol-specific metrics (ingress and egress bytes). Harness and Doit do not support ingestion of external telemetry, relying exclusively on CSP billing and metadata.

Metric source diversity varies considerably. The most flexible platforms, such as Finout and Vantage.sh, ingest data from multiple telemetry sources (Prometheus, Datadog, Snowflake, CloudWatch, S3, CSV, and CSP billing). CloudZero supports a broad set of integrations, including Datadog and Snowflake. Conduktor is limited to Kafka traffic metrics; Harness and Doit are limited to billing data and metadata from major public clouds.

Multi-metric support is not widely documented; only Finout and Vantage.sh explicitly allow ingestion of several metrics. However, none of the reviewed tools clearly provides support for combining multiple metrics in a single allocation model; allocation is generally based on a single selected metric.

Metric weighting is notably absent in all tools, which is a key gap to reflect the relative impact of multiple usage metrics in the cost allocation.

All tools support at least one form of shared cost allocation: equal split and fixed percentage are universally available, while CloudZero, Finout, and Vantage.sh also offer telemetry-based proportional split.

Finally, all tools offer a built-in UI for cost visualization and dashboarding.

In summary, while the commercial landscape offers advanced metadata-driven cost attribution, integration with multiple data sources, and flexible shared cost strategies, it is still limited by the lack of multi-metric weighting.

3.4 Conclusion

The systematic review of both academic literature and industry solutions reveals several key findings regarding cost allocation in shared cloud infrastructure environments.

The literature emphasizes that successful cost allocation systems must balance multiple factors, as identified by Baars *et. al* [7]. These include accuracy, transparency, fairness, and predictability, among others. These factors can be categorized into two main dimensions: effectiveness (how well the system achieves cost awareness) and acceptability (stakeholder acceptance of the allocation process).

Current industry solutions demonstrate varying approaches to addressing these challenges. Most solutions support multiple cloud providers and implement some form of proportional cost allocation, either through manual configuration or automated telemetry-based approaches. Tools

like CloudZero [10], Finout [19], and Doit [16] have introduced tagging mechanisms and custom dimensioning capabilities to organize and allocate costs across different organizational units.

However, the analysis reveals two significant gaps in the current landscape of cost allocation solutions.

First, while existing solutions collect telemetry data and support basic proportional allocation, none provide a mechanism for combining multiple metrics with different weights into a unified cost function. Current approaches typically rely on single metrics, lacking the ability to define complex cost functions that could more accurately reflect the true impact of resource usage. This limitation prevents organizations from implementing nuanced cost allocation strategies that consider multiple aspects of resource consumption with varying degrees of importance.

The second major gap lies in how current solutions handle inter-team resource sharing patterns, particularly in scenarios where one team provides services to others. This gap is particularly significant in modern organizational structures where teams provide shared services to multiple other teams. The inability to account for these service-provider relationships in cost allocation can lead to unfair distribution of costs and potentially discourage teams from providing shared services that could benefit the organization as a whole.

These findings suggest that while existing solutions provide valuable capabilities for cost allocation in shared cloud infrastructure, there remains a significant opportunity for advancement.

Chapter 4

Problem Statement

In this chapter, we present the problem statement that motivates this dissertation. In Section 4.1, we describe the limitations of current cost allocation approaches in shared cloud infrastructure environments, identifying key challenges that arise in practice. Section 4.2 outlines the desiderata that a solution must fulfill to address these challenges effectively. In Section 4.3, we formulate the central hypothesis guiding this work. Section 4.4 derives the main research questions from the hypothesis. Finally, Section 4.5 introduces the proposed solution, briefly describing its core components and objectives.

4.1 Problem

As organizations scale, cloud infrastructure becomes both a critical enabler and a complex liability. Most large enterprises adopt shared cloud components. This multi-tenant environment introduces an accounting challenge: how to fairly and accurately allocate shared infrastructure costs to consuming teams.

The three most commonly used allocation strategies—even splitting, fixed proportional allocation, and variable proportional allocation—offer progressively improved fidelity, with variable proportional allocation based on telemetry data generally considered the most accurate, as it leverages real-time telemetry data to reflect actual resource usage.

All of the industry tools detailed in Chapter 3 implement at least one of these allocation strategies. However, those that support variable proportional allocation typically rely on a single telemetry metric, thereby failing to account for the differentiated cost impact of distinct resource types. This is a critical limitation, given that real-world infrastructure usage is inherently heterogeneous: some services are compute-intensive, others are bandwidth-heavy, and some are primarily disk-bound. Relying on a single metric leads to inaccurate cost attribution.

Moreover, associating resources with logical units (e.g., teams, projects, departments) is itself a non-trivial problem. In current tools, this mapping is most commonly achieved through tagging, the use of metadata keys and values applied to resources. Cost attribution mechanisms then rely

on combinations of these tags to group usage under specific entities. As such, the accuracy of cost attribution is directly dependent on the quality and granularity of the mapping strategy.

Finally, in modern organizations, it is common for teams to develop and operate services that are consumed by other teams. When cost allocation fails to recognize these producer-consumer relationships, shared costs may be misattributed to the provider teams. This not only results in an unfair financial burden but also disincentivizes the development and maintenance of shared services. A behavior that undermines scalability, efficiency, and reuse across the organization.

4.2 Desiderata

To address the identified limitations, an effective solution should satisfy the following desiderata:

- D1 Multi-Metric Cost Modeling** The framework must support multiple telemetry metrics as the basis for cost allocation. Rather than attributing costs based on a single usage metric, the system must be capable of aggregating multiple metrics, each contributing to the final allocation outcome.
- D2 Differentiated Metric Impact** The framework must provide a mechanism to reflect the unequal cost impact of different telemetry metrics in cost allocation. Resource types such as CPU, memory, storage, and network bandwidth do not contribute equally to infrastructure costs.
- D3 Resource to Logical Unit Mapping** Accurate cost attribution requires the ability to associate resource usage with logical organizational entities (e.g., teams, departments, or projects). The system must support robust mapping using available metadata.
- D4 Support for Inter-Team Redistribution** To model real-world service relationships, the framework must support the redistribution of shared costs. It must allow for the specification of consumer-provider relationships, such that costs can be partially or fully reassigned from provider teams to consuming teams based on actual usage patterns.
- D5 Transparency and Explainability** The system must offer transparent and auditable cost attribution. Every allocation decision must be traceable and explainable. This is essential for building trust in the system and enabling teams to understand and validate allocation results.

4.3 Hypothesis

In this dissertation, the work presented in the following chapters aims to validate the following hypothesis:

“In organizations that operate shared infrastructure, a framework that allocates costs based on multiple metrics, incorporates the differentiated cost impact of resource types, and supports inter-team cost redistribution will improve fairness.”

By "*organizations that operate shared cloud infrastructure*", we refer to companies with multiple teams consuming common resources, such as data platforms, messaging systems, or internal developer tools, where accurate attribution of shared costs is financially critical.

By "*allocates costs based on multiple metrics*", we mean that the framework considers several types of resource consumption (e.g., CPU, memory, bandwidth, storage) rather than relying on a single metric to determine how costs are distributed.

In "*incorporates the differentiated cost impact of resource types*", we refer to the ability of the system to account for the fact that not all resource metrics contribute equally to infrastructure costs; there must be a mechanism to model this imbalance appropriately.

Lastly, by "*supports inter-team cost redistribution*", we refer to the system's ability to model consumer-provider relationships within the organization, redistributing costs when one team provides a service that is used by others.

To sum up, this thesis aims to develop and validate a cost allocation framework for shared infrastructure environments. By leveraging multiple resource metrics, accounting for their differentiated cost impact, and enabling inter-team cost redistribution, the proposed framework seeks to improve the fairness of cost attribution across teams.

4.4 Research Questions

Based on the hypothesis above, the following research questions were created:

- **RQ1:** *How can telemetry data be used to allocate infrastructure costs in shared environments?* This question investigates the role of usage data in accurately distributing shared costs. It seeks to explore how telemetry can serve as the foundation for cost allocation.
- **RQ2:** *How can resource usage be accurately mapped to organizational entities (e.g., teams, departments, or projects) using available metadata?* This question addresses the challenge of correlating low-level resource consumption with high-level organizational structures, which is crucial for team-level accountability.
- **RQ3:** *What is the impact of considering multiple metrics with differentiated cost impact on the fairness of cost allocation?* Since modern workloads utilize heterogeneous resources, this question explores how integrating multiple usage dimensions (e.g., CPU, memory, network) and assigning them appropriate weights improves the fairness of cost allocation.
- **RQ4:** *What approaches can be used to determine the relative contribution or importance of each metric in the allocation model?* This question focuses on how to identify appropriate weights for different resource metrics, ensuring that each metric's influence aligns with its actual cost impact in the infrastructure.
- **RQ5:** *How can the cost allocation be extended to incorporate inter-team service relationships and incentivize efficient, collaborative infrastructure usage?* This question explores

mechanisms for recognizing producer-consumer dynamics in shared environments and redistributing costs to promote cooperation, service reuse, and fair cost burden distribution.

- **RQ6:** *What validation techniques and evaluation metrics can be applied to assess the fairness of the proposed cost allocation framework?* The final question focuses on the empirical assessment of the framework. It investigates which methods can verify whether the allocation approach meets fairness.

4.5 Proposed Solution

The solution proposed in this dissertation is designed to address the challenges outlined in the problem statement and fulfill the stated hypothesis. It consists of the development of a cost allocation framework capable of operating in shared infrastructure environments. The framework will support multi-metric cost modeling, account for the differentiated cost impact of resource types, enable inter-team cost redistribution, and generate comprehensive reporting to ensure transparency of the results.

The implementation will include the necessary software components to ingest telemetry data, map resource usage to organizational units using metadata, and apply the proposed allocation logic.

Chapter 5

Solution

In this chapter, we present the proposed solution to the cost allocation challenges identified in the previous chapter. Section 5.1 outlines the core principles behind the framework, highlighting its cloud-agnostic design, multi-dimensional cost modeling, and support for logical unit mapping and inter-team redistribution. Section 5.2 defines the formal model that governs the cost allocation logic, including both the base allocation and discount mechanisms. Section 5.3 describes the practical implementation of the framework, detailing its architecture and configuration.

5.1 Core Principles

The proposed solution addresses the key limitations identified. Most existing cost allocation tools are tightly coupled with specific cloud providers and rely heavily on provider native billing mechanisms or tagging systems. In contrast, this framework is designed to be cloud-agnostic, emphasizing a technology-oriented approach that can adapt to diverse infrastructure environments.

At its core, this solution implements a proportional cost distribution strategy, where costs are allocated based on each logical unit's consumption of shared resources. Crucially, this allocation is driven by telemetry data, enabling dynamic and data-informed cost assessments.

Unlike static or single-metric models, the framework introduces a multi-dimensional weighting mechanism. For example, in a messaging system, you might weight network usage higher than storage due to bandwidth billing costs. Each dimension (e.g., network, storage, compute) represents a type of resource usage and contains a set of associated metrics relevant to that category. These dimensions are each assigned a configurable weight to reflect their relative cost impact.

A foundational requirement for cost allocation is the correlation of resource consumption to responsible logical units. This framework achieves that through metric label analysis using configurable regular expressions. Each metric label (e.g., "kn.team1.topic1") is parsed to extract logical units (e.g., "team1"), enabling mapping to organizational units. This method allows for flexible and consistent association of telemetry data with the entities accountable for costs.

The architecture is intentionally modular: it can accommodate any system that emits labeled metrics, as long as those labels can be extracted and mapped to logical units. As a result, the same foundational engine can be reused across environments with only configuration changes.

In addition, the framework addresses a significant gap observed in existing solutions: the inability to fairly represent consumer-provider relationships in shared infrastructure scenarios. Identified tools fail to capture these interactions, leading to imbalanced or inaccurate cost allocations. To solve this, the framework introduces a discount mechanism — a configurable, rule-based system that allows one logical unit's allocated cost to be partially shifted to another. This enables more accurate reflection of service dependencies and organizational dynamics, such as rewarding service providers or charging back consumers proportionately.

5.1.1 Definition of Logical Units

In the context of this framework, a logical unit represents any unit that we want to allocate the costs to, such as a team, service, department, or application. Logical units are the ultimate recipients of cost allocations and serve as the anchors for financial accountability.

5.2 Formal Cost Allocation Model

This section presents the formal model underpinning the proposed framework's cost allocation logic. The objective is to define a mathematical structure capable of distributing a total cost across logical units based on their proportional consumption of shared resources. The formulation is intentionally abstract and general, ensuring its applicability across different technologies and metric sources as long as telemetry data can be collected and correlated with responsible entities.

Although the practical implementation of the system performs the calculation in a bottom-up fashion, starting from raw metric values, this section follows a top-down reasoning process, as it more clearly reflects the conceptual design and motivation behind each model layer.

The formal model is divided into two main parts: (1) base cost allocation, which defines how total cost is proportionally distributed based on usage; and (2) discount application, which introduces a mechanism for adjusting the initial allocations to reflect shared service relationships or incentivize desired behaviors.

5.2.1 Base Cost Allocation

The base cost allocation begins with the highest level concept (total cost to be allocated) and progressively decomposes each model layer until the metric-level granularity is reached.

5.2.1.1 Total Cost

Since we are following a top-down approach, it only makes sense for the starting point to be the total cost to be allocated (TC). The fundamental principle is that the sum of the cost of all logical

units must equal TC as represented in equation 5.1. This ensures completeness in the allocation process: every logical unit is accounted for.

$$TC = \sum_{lu \in LU} C_{lu} \quad (5.1)$$

Where:

- TC is the total cost to be allocated
- LU is the set of all logical units
- C_{lu} is the cost allocated to a logical unit lu

5.2.1.2 Cost of a single logical unit

As discussed before, the proposed solution follows a proportional allocation based on usage. With this in mind, each logical unit receives a portion of the total cost based on its normalized usage factor. Therefore, the cost of a logical unit is equal to the usage factor for this logical unit times the total cost to be allocated, as shown in equation 5.2.

$$C_{lu} = U_{lu} \times TC \quad (5.2)$$

Where:

- U_{lu} is the usage factor for logical unit lu (between 0 and 1)

With:

$$\sum_{lu \in LU} U_{lu} = 1 \quad (5.3)$$

5.2.1.3 Usage of a logical unit

The usage factor of a logical unit is modeled as a composition of multiple dimensions, where each dimension represents a distinct category of resource consumption that contributes to the overall cost. Typical dimensions include, for example, network and storage. Since the cost impact of each resource type may vary, every dimension is assigned a weight that quantifies its relative importance in the total cost structure. The overall usage of a logical unit is then computed as the weighted sum of its usage across all dimensions as represented in equation 5.4.

$$U_{lu} = \sum_{d \in D} w_d \times U_{lu,d} \quad (5.4)$$

Where:

- D is the set of dimensions
- w_d is the weight of dimension d

- $U_{lu,d}$ is the usage of logical unit lu in dimension d

With:

$$\sum_{d \in D} w_d = 1 \quad (5.5)$$

5.2.1.4 Usage of a logical unit in a dimension

As introduced previously, each dimension represents a category of resource usage. Within a dimension, usage is further broken down into one or more metrics, where each metric captures a specific measurable aspect of that dimension (e.g., byte throughput, log size). To account for the varying significance of individual metrics, each metric is assigned a weight that expresses its contribution to the dimension as a whole. Consequently, the usage of a logical unit in a given dimension is computed as the weighted sum of its usage across all associated metrics, as formally defined in equation 5.6.

$$U_{lu,d} = \sum_{m \in M_d} w_{d,m} \times U_{lu,d,m} \quad (5.6)$$

Where:

- M_d is the set of metrics for dimension d
- $w_{d,m}$ is the weight of metric m in dimension d
- $U_{lu,d,m}$ is the usage of logical unit lu for metric m in dimension d

With:

$$\sum_{m \in M_d} w_{d,m} = 1 \quad (5.7)$$

5.2.1.5 Aggregation from Labels to Logical Units

The relationship between logical units and telemetry data is established through metric labels, which are parsed using configurable regular expressions. Each metric label is expected to contain a value that can be extracted to identify the responsible logical unit.

For instance, consider a label value such as "kn.team1.topic1". A user-defined regex pattern might extract the substring "team1" as the logical unit. This extraction logic is entirely configurable, allowing the system to adapt to different naming conventions or labeling strategies.

Table 5.1: Examples of metric labels and how regex patterns extract logical units.

Metric Label	Regex Pattern	Group	Extracted Logical Unit
kn.team1.topic1	kn(.*)	1	team1
svc.team2.queue45	svc(.*)	1	team2
metrics.devops.cpu	metrics(.*)	1	devops

Once labels are mapped to logical units, each unit's usage for a given metric is calculated by aggregating the metric values of all associated labels, as shown in Equation 5.8:

$$U_{lu,d,m} = \sum_{l \in L_{lu}} U_{l,d,m} \quad (5.8)$$

Where:

- L_{lu} is the set of labels that are associated with logical unit lu
- $U_{l,d,m}$ is the usage of label l for metric m in dimension d

5.2.1.6 Metric Results and Relevant Label Keys

Before jumping into the usage of a label, we need to better understand the composition of a metric result. The result for a metric is an array of tuples, where each is represented as:

$$r = (\{(lk_1, lv_1), (lk_2, lv_2), \dots, (lk_n, lv_n)\}, Q_r) \quad (5.9)$$

Where:

- $L_r = \{(lk_i, lv_i)\}$ is a set of label key-value pairs.
- Q_r is the metric value returned for that label combination.

For each metric, only a single label key is used for allocation, defined as the **relevant label key** ($RLK_{d,m}$). The label value corresponding to this key is extracted using:

$$RelevantLabelValue(L_r, RLK_{d,m}) = lv, \text{ such that } (lk = RLK_{d,m}, lv) \in L_r \quad (5.10)$$

When multiple metric results contain the same relevant label value, their values are aggregated by summing Q_r values.

5.2.1.7 Label Usage

The usage of a relevant label value for a specific metric is the proportion of the metric's value for that label to the sum of all metric values, as represented in equation 5.11.

$$U_{l,d,m} = \frac{Q_{l,d,m}}{\sum_{l' \in L} Q_{l',d,m}} \quad (5.11)$$

Where:

- $Q_{l,d,m}$ is the metric value for label l , metric m in dimension d

5.2.1.8 Base Allocation Conclusion

The base allocation model described above provides a layered and composable approach to attributing shared infrastructure costs. Beginning with the total cost TC , it systematically decomposes the allocation through logical units, dimensions, metrics, and label-level usage.

For clarity, the full composition of the cost allocated to a logical unit lu can be summarized as follows:

$$C_{lu} = TC \times \sum_{d \in D} w_d \times \sum_{m \in M_d} w_{d,m} \times \left(\sum_{l \in L_{lu}} \frac{Q_{l,d,m}}{\sum_{l' \in L} Q_{l',d,m}} \right) \quad (5.12)$$

This expression shows that the cost allocated to each logical unit is a function of the total cost, weighted dimensional and metric contributions, and the normalized usage derived from metric results.

The model thus forms the foundational layer of the proposed cost allocation model.

5.2.2 Discount Mechanism

The goal of the discount mechanism is to shift a portion of the cost from one or more logical units, identified by discounted labels, to others. The total cost remains unchanged; only its distribution is adjusted according to user-defined discount rules.

5.2.2.1 Discount Attributes

Each discount rule d is defined by the following attributes:

- A **trigger metric** m , which provides the metric values used for determining and distributing the discount.
- A **condition** θ , a predicate applied to the metric value to determine whether a label qualifies for discounting.
- A **discounted label key** k_d , used to extract the label value for the discounted entity.
- A **penalized label key** k_p , used to extract the penalized label value, or the special value `_all` to denote redistribution across all other logical units.
- A **discount percentage** $p \in (0, 1)$, indicating the fixed percentage to be reallocated per qualifying discounted label.
- A **reallocation strategy**, either *proportional* or *equal*, specifying how to distribute the discount among penalized units.

Let R be the set of metric results returned for the trigger metric m . As detailed before, each result $r \in R$ is composed of:

- A set of label pairs $L_r = \{(lk_i, lv_i)\}$,
- A scalar metric value Q_r .

From each result r , we extract:

- The **discounted label value**: $l_d = L_r[k_d]$,
- The **penalized label value**: $l_p = L_r[k_p]$, if $k_p \neq \text{all}$,
- The **metric value**: Q_r .

5.2.2.2 Identify Eligible Discounted Labels

Let LD be the set of all discounted label values found in the metric results. The subset of those that the metric value for that label ($Q_{r,l}$) satisfies the condition θ forms the eligible set LDE :

$$LDE = \{l \in LD \mid \theta(Q_{r,l}) = \text{true}\} \quad (5.13)$$

5.2.2.3 Redistribution

After identifying the eligible discounted labels LDE , the framework performs two steps:

1. Applies the discount to the logical unit associated with each discounted label.
2. Redistributes the discounted amount p to one or more penalized logical units.

Let lu_d be the logical unit associated with the discounted label $l_d \in LDE$. The total cost allocated in the base allocation to lu_d is reduced by p , reflecting the incentive or reward triggered by the discount rule.

The discounted amount p is then redistributed across other logical units, depending on the configuration of each discount rule, specifically the penalized label key k_p and the chosen redistribution strategy.

Case 1: Penalized label key $k_p \neq \text{all}$

In this case, the discount is redistributed only among the logical units associated with specific penalized label values found alongside each discounted label in the trigger metric results.

1. For each discounted label value $l_d \in LDE$, the framework identifies all penalized label values l_p that appear in the same metric result as l_d . These label values form the set:

$$LP_{l_d} = \{l_p \mid \exists r \in R \text{ such that } L_r[k_d] = l_d \text{ and } L_r[k_p] = l_p\} \quad (5.14)$$

LP_{l_d} represents the set of penalized labels associated with the discounted label l_d .

2. For each penalized label $l_p \in LP_{l_d}$, the corresponding metric value Q_{l_p} is extracted from the same metric result. This value is used to guide the redistribution, depending on the strategy.

The total cost to be redistributed for each l_d is fixed at p . This value is then reallocated to logical units derived from l_p using one of two strategies:

- **Proportional strategy:**

This strategy distributes the discount in proportion to the penalized labels' metric result, as measured by Q_{l_p} . The total value is calculated as:

$$V = \sum_{l_p \in LP_{l_d}} Q_{l_p} \quad (5.15)$$

Each logical unit lu_p corresponding to label l_p receives a portion:

$$\Delta_{lu_p} = p \cdot \frac{Q_{l_p}}{V} \quad (5.16)$$

This ensures that logical units that contribute more to the measured metric bear a larger share of the reallocated cost.

- **Equal strategy:**

In this strategy, the discount is divided equally among the penalized units, regardless of their individual metric values. If there are $N = |LP_{l_d}|$ penalized label values, each logical unit receives:

$$\Delta_{lu_p} = \frac{p}{N} \quad (5.17)$$

This approach is useful when all penalized units should be treated equally.

Case 2: Penalized label key $k_p = _a11$

In this configuration, the redistribution is not based on any specific penalized label key. Instead, the discount amount p , applied to the logical unit lu_d associated with the discounted label l_d , is redistributed equally across all other logical units in the system.

The penalized set of logical units in this case is not derived from metric labels. Instead, it is explicitly defined as:

$$LU_p = LU \setminus \{lu_d\} \quad (5.18)$$

That is, all logical units *except* the discounted one.

Let $N = |LU_p|$ be the number of penalized units. Then, the cost increase applied to each penalized logical unit $lu_p \in LU_p$ is:

$$\Delta_{lu_p} = \frac{p}{N} \quad (5.19)$$

This strategy is particularly useful for rewarding specific behaviors, where one unit is rewarded (discounted) and the remaining percentage is absorbed by all others.

5.2.2.4 Discount Mechanism Conclusion

The discount mechanism enriches the base allocation model by introducing flexible, rule-based redistribution of costs. By allowing certain logical units to be rewarded or penalized based on usage conditions and metric-driven logic, the model accounts for real-world organizational dynamics such as service consumption patterns, shared infrastructure responsibilities, or incentive alignment. The mechanism preserves the integrity of the total cost while enabling nuanced redistribution through well-defined, configurable strategies.

5.3 Implementation

This section details the practical implementation of the cost allocation framework, focusing on its integration with existing telemetry infrastructure, its architecture, and configuration. We first describe how Prometheus is used for metric ingestion and result publication. Next, we present a high-level architectural overview, detailing the main components of the framework. We then explore the structure of the declarative configuration file, which maps directly to the formal model. Finally, we discuss the main algorithmic components: the base allocation and the discount mechanism.

5.3.1 Prometheus

Prometheus [34] is an open-source systems monitoring and alerting toolkit designed for reliability and scalability in modern infrastructure environments.

Prometheus operates using a **pull-based data collection model**, in which it periodically sends HTTP requests to predefined endpoints, referred to as targets, to retrieve metrics. Each target must expose a `/metrics` HTTP endpoint that returns data in a plain-text exposition format defined by the Prometheus specification. This model allows Prometheus to control the frequency of data collection.

5.3.1.1 Exporters and Targets

A target is any endpoint that exposes metrics in the Prometheus exposition format. In practice, many systems do not natively support Prometheus instrumentation. To address this, Prometheus uses exporters.

An exporter is an auxiliary service or binary that runs alongside the application or infrastructure component to be monitored. Its primary function is to expose metrics in the Prometheus exposition format, typically by translating metrics from proprietary, domain-specific, or otherwise incompatible formats into a structure that Prometheus can ingest. This approach allows

Prometheus to collect metrics from a wide variety of technologies, such as databases, messaging systems, and operating systems, even if they do not natively support Prometheus integration.

5.3.1.2 Time-Series Database

All scraped metrics are stored in Prometheus' internal **time-series database** (TSDB). A time-series database is optimized for storing and querying measurements indexed over time. Each time series in Prometheus is uniquely identified by its metric name and a set of labels (key-value pairs) that provide contextual metadata.

The TSDB supports high ingestion rates, efficient compression, and retention policies, making it well-suited for real-time monitoring and historical trend analysis.

5.3.1.3 Prometheus in This Framework

In the context of this framework, Prometheus plays a dual role:

- **As a scraper and telemetry source**, Prometheus collects infrastructure-level metrics by scraping various exporters associated with the target technology. These metrics serve as input for cost allocation calculations.
- **As a consumer of cost allocation output**, Prometheus also scrapes the Cost Allocation Engine, which exposes computed allocation results through its own metrics endpoint, functioning as a custom Prometheus exporter.

This dual integration creates a feedback loop: Prometheus not only supplies the telemetry used in the cost computation but also stores and makes available the resulting cost allocations for visualization.

5.3.2 Architecture High Level Overview

The architecture of the cost allocation framework is organized into distinct layers that separate metric collection, cost allocation processing, and reporting. It relies on established monitoring infrastructure for metric acquisition and uses modular internal components for extensible metric processing and cost allocation. At the foundation, a set of Prometheus exporters runs alongside the target technology. These exporters expose domain-specific metrics in the Prometheus exposition format.

Prometheus serves as the central telemetry aggregator. It periodically scrapes all registered exporters, building a time-series database of metrics. This data foundation is then used by the cost allocation engine.

Within the cost allocation engine, the metric scrapers module is responsible for querying Prometheus or other sources for relevant metrics. These scrapers are extensible, enabling support for additional sources simply by implementing the required abstract interface. The scrapers supply all needed telemetry to the core allocation logic.

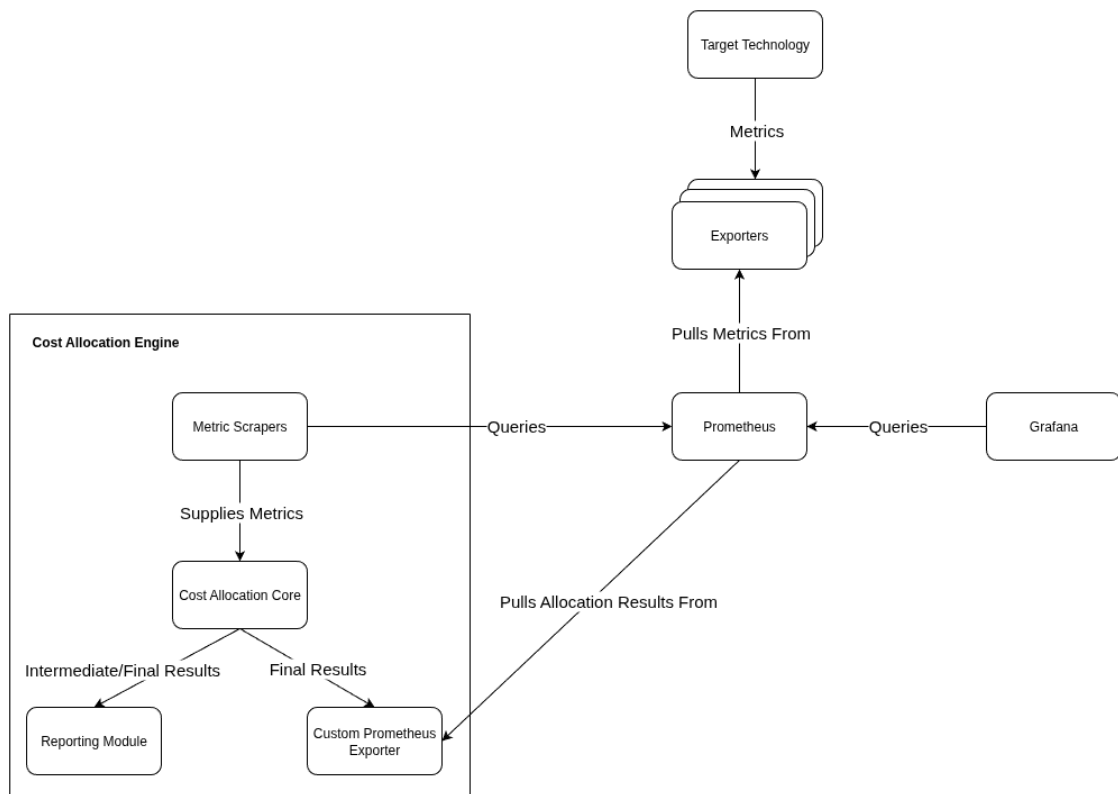


Figure 5.1: High-level Architecture Diagram

Once the allocation is completed, results are exported by the engine using a custom Prometheus exporter. This exporter exposes the cost allocation outcomes as metrics at an HTTP endpoint, which is again scraped by Prometheus at regular intervals. In this manner, Prometheus not only supplies the engine with the input telemetry but also consumes and stores the computed allocation results. This forms a closed feedback loop where raw usage data is transformed into financial observability, with all results being accessible via the same time-series monitoring infrastructure.

A key internal component of the engine is the reporting module. After each allocation run, this module generates detailed, human-readable reports documenting all intermediate and final results, including metric inputs, intermediate aggregation steps, application of discounts, and the final allocation. These reports serve as an audit trail, supporting traceability and explainability of the cost allocation process. Reports are persisted as files and can be used for compliance checks or for deeper analysis.

For visualization, Grafana [25] is connected to Prometheus and serves as an interactive dashboard for querying and displaying both cost allocation results and metrics from the target technology. This enables both real-time monitoring and historical analysis of allocation results.

5.3.3 Configuration File Specification

The configuration file at the core of this cost allocation framework serves as the declarative blueprint for the entire allocation process. This section provides a description of the configuration file, detailing each of its components. To support both clarity and extensibility, the configuration file is divided into three well-defined sections, each serving a specific purpose within the allocation framework: Logical Unit Extraction and Labeling, Dimensions and Metrics, and Discount Rules. All sections are correlated with the formal model detailed in Chapter 5 Section 5.2.

5.3.3.1 Logical Unit Extraction and Labeling

As detailed before, the correlation between logical units and resources is made through regex extraction in metric labels. The configuration file expresses this through two related settings: *logical_unit_label* and *logical_unit_relation*. The *logical_unit_label* designates the label key under which the allocation results will be indexed (for instance, "team" in a multi-team environment). The *logical_unit_relation* is an object specifying how to extract the relevant logical unit from the raw metric labels using a regular expression. It's composed of two fields: *extract_regex* that holds the actual regular expression and *group_num* that specifies which capture group of the regular expression contains the logical unit's identifier. This configuration is used to extract the logical unit from all identified labels.

5.3.3.2 Dimensions and Metrics

At the core of the cost allocation process is the notion of dimensions. Every dimension is assigned a weight, which quantifies its impact in the total cost formula. Within each dimension, there is a list of metrics. Each metric has several defining attributes:

- **weight:** The impact of the metric in its parent dimension. All metric weights in a dimension must sum to one.
- **name:** The extract metric or query name.
- **relevant_label:** The label key, among the metrics label set, that is relevant for the cost allocation.

5.3.3.3 Discount rules

The discounts section in the configuration file is composed of a set of discount rules. Each rule is an object describing how a portion of the cost should be redistributed among logical units under certain conditions, as explained in Chapter 5 Section 5.2.2. The fields in each discount rule are:

- **name:** The identifier of the discount.
- **trigger:** The metric or query that provides the metric values used for determining and distributing the discount.

- **discounted_label:** A key used to extract the label value for the discounted entity in the metric results labels.
- **discounted_condition:** A comparison operator and value that define when the discount applies according to the metric result.
- **penalized_label:** A key used to extract the penalized label value, or the special value `_all` to denote global redistribution across all other logical units.
- **reallocation_strategy:** Specifies whether the reallocated cost should be split proportionally to metric values or equally among penalized units.
- **discount_percentage:** The percentage to be reallocated per label.

5.3.4 Cost Allocation Logic

The cost allocation logic consists of two sequential stages: the *Base Allocation* and the *Discount Mechanism*. The base allocation algorithm attributes costs to logical units in proportion to their weighted usage of various metrics and resource dimensions. The output of this stage is a mapping from logical units to their allocated values. The discount mechanism then refines these allocations by applying user-defined rules that redistribute costs among logical units according to specified conditions and redistribution strategies. This process is realized as a pipeline of algorithms: eligibility and extraction of applicable discounted and penalized labels, followed by one of three redistribution strategies: global, proportional, or equal, applied according to each rule's configuration.

5.3.4.1 Base Allocation

The *Base Allocation* algorithm starts by iterating through all dimensions defined, for each dimension, it iterates over all associated metrics. Metric results are first collected; each result is characterized by a set of labels and an associated numeric value representing measured usage. For each metric, a relevant label key is defined in the configuration. This relevant label key specifies which key among the set of metric labels is used to group and aggregate results for cost allocation. Only the value corresponding to this relevant label key is considered in the allocation process; all other label keys present in the metric are ignored. The algorithm then computes the sum of all metric values for the current metric, which serves as a normalization constant for that metric. Each individual metric value is divided by this sum, yielding a normalized value that represents the proportion of the metric's total usage attributable to the given label. This step ensures that the impact of each label is proportional to its contribution relative to all other labels for the same metric.

Once normalization is complete, the algorithm applies both the metric weight and the dimension weight to each normalized value. The product of these weights and the normalized value results in the final weighted contribution of each label for that metric and dimension. This value is

then accumulated into a map structure keyed by label, which collects the weighted contributions across all metrics and dimensions.

After all metric results have been processed, the algorithm groups these label contributions according to logical units. The mapping from label to logical unit is performed using a regular expression defined in the system configuration. Each logical unit receives the sum of all contributions from labels associated with it. Labels that do not match the regex expression are grouped under an artificial category (“others”).

The output of the base allocation step is a mapping from logical units to their total accumulated usage across all specified dimensions.

Algorithm 1 Base Allocation

```

1: results_by_label  $\leftarrow \{\}$ 
2: for all dimension in config do
3:    $w_d \leftarrow \text{dimension.weight}$ 
4:   for all metric in dimension.metrics do
5:      $w_m \leftarrow \text{metric.weight}$ 
6:      $R \leftarrow \text{query\_metric}(\text{metric.name})$ 
7:      $S \leftarrow \sum_{r \in R} r.\text{value}$ 
8:     for all result in  $R$  do
9:        $\text{label\_value} \leftarrow \text{result.labels}[\text{metric.relevant\_label}]$ 
10:       $\text{normalized} \leftarrow \text{result.value} / S$ 
11:       $\text{weighted} \leftarrow \text{normalized} \times w_m \times w_d$ 
12:       $\text{results\_by\_label}[\text{label\_value}] += \text{weighted}$ 
13:    end for
14:  end for
15: end for
16:
17: results_by_logical_unit  $\leftarrow \{\}$ 
18: for all (label_value, value) in results_by_label do
19:    $lu \leftarrow \text{extract\_logical\_unit}(\text{label\_value})$ 
20:   if  $lu$  is None then
21:      $lu \leftarrow \text{“others”}$ 
22:   end if
23:    $\text{results\_by\_logical\_unit}[lu] += \text{value}$ 
24: end for

```

5.3.4.2 Discount Mechanism

The discount mechanism is responsible for refining the base allocation by applying a series of user-defined rules that adjust cost allocations among logical units according to specific criteria and redistribution strategies. The process is structured as a pipeline of algorithms, each building

upon the results of the previous step. The process begins with the *Discount Eligibility and Extraction* algorithm, which examines each discount rule and its trigger metric. For every rule, it checks which metric results satisfy the rule's condition and collects the relevant discounted and penalized labels, along with their metric values. These are grouped so that, for each eligible discounted label, all associated penalized labels and values are organized together. This structure is then used as input for the redistribution algorithms that follow. Depending on the configuration of each rule, one of three redistribution algorithms is then applied: *Global Redistribution*, *Proportional Strategy*, or *Equal Strategy*. The *Global Redistribution* algorithm handles cases where the discount is to be shared among all logical units, the *Proportional Strategy* and *Equal Strategy* algorithms redistribute discounts among targeted logical units either in proportion to their metric values or equally, respectively. Each of these algorithms updates the allocation results from the base allocation step.

1. Discount Eligibility and Extraction In the first step of the discount mechanism, the algorithm iterates through each configured discount rule. For each rule, it queries the relevant trigger metric and processes each result to determine eligibility. Eligibility depends on a condition specified in the rule, which defines a comparison operator and a threshold. For every result that meets the condition, the algorithm extracts the discounted label value. If the penalized label is not set to *_all*, it also extracts the penalized label value. All penalized labels and their metric values are grouped under their associated eligible discounted label, creating a mapping from each discounted label value to a collection of penalized label values and their metric values. This grouping is essential for the redistribution process in subsequent steps.

Algorithm 2 Discount Eligibility and Extraction

```

1: for all discount_rule in config.discounts do
2:   trigger_results  $\leftarrow$  query_metric(discount_rule.trigger)
3:   eligible_discounts  $\leftarrow$  {}
4:   for all result in trigger_results do
5:     discounted_label  $\leftarrow$  result.labels[discount_rule.discounted_label]
6:     value  $\leftarrow$  result.value
7:     if discount_rule has condition then
8:       if not check_condition(value, discount_rule.condition) then
9:         continue
10:      end if
11:    end if
12:    if discounted_label not in eligible_discounts then
13:      eligible_discounts[discounted_label]  $\leftarrow$  []
14:    end if
15:    if discount_rule.penalized_label  $\neq$  “_all” then
16:      penalized_label  $\leftarrow$  result.labels[discount_rule.penalized_label]
17:      eligible_discounts[discounted_label].append((penalized_label, value))
18:    end if
19:  end for
20: end for

```

2. Global Redistribution If the discount rule specifies the global redistribution case, where the penalized label is set to *_all*, the algorithm reduces the allocation of the discounted logical unit by the specified discount value. The total discounted amount is then redistributed equally across all other logical units in the system, with the exception of the discounted unit itself.

Algorithm 3 Global Redistribution (*penalized_label* = *_all*)

```

1: for all (discounted_label, _) in eligible_discounts do
2:   discounted_unit  $\leftarrow$  extract_logical_unit(discounted_label)
3:   discount_value  $\leftarrow$  discount_rule.discount_percentage
4:   results_by_logical_unit[discounted_unit]  $- =$  discount_value
5:   penalized_units  $\leftarrow$  { lu | lu in results_by_logical_unit and lu  $\neq$  discounted_unit }
6:   for all penalized_unit in penalized_units do
7:     results_by_logical_unit[penalized_unit]  $+ =$  discount_value / |penalized_units|
8:   end for
9: end for

```

3. Proportional Strategy When a discount rule uses a proportional reallocation strategy, the algorithm iterates through all eligible discounted labels. For each extracts the corresponding dis-

counted logical unit and subtracts the discount value from its allocation in the base allocation results. It then considers all penalized labels associated with this discounted label, extracting the penalized logical unit and its metric value for each. The algorithm computes the sum of all penalized metric values and, for each penalized logical unit, increases its allocation value by a share of the discount proportional to its metric value divided by the total sum.

Algorithm 4 Proportional Strategy

```

1: for all (discounted_label, penalized_list) in eligible_discounts do
2:   discounted_unit  $\leftarrow$  extract_logical_unit(discounted_label)
3:   discount_value  $\leftarrow$  discount_rule.discount_percentage
4:   results_by_logical_unit [discounted_unit]  $- =$  discount_value
5:   S  $\leftarrow$  0
6:   for all (_, value) in penalized_list do
7:     S  $\leftarrow$  S + value
8:   end for
9:   for all (plabel, value) in penalized_list do
10:    penalized_unit  $\leftarrow$  extract_logical_unit(plabel)
11:    results_by_logical_unit[penalized_unit]  $+=$  discount_value  $\times$  (value/S)
12:   end for
13: end for

```

4. Equal Strategy For discount rules that use the equal reallocation strategy, the algorithm processes each eligible discounted label by identifying the corresponding discounted logical unit and subtracting the discount value from its allocation. It then considers all penalized labels associated with this discounted label, extracting the penalized logical unit for each. The algorithm determines the total number of penalized logical units and, for each one, increases its allocation by an equal share of the discount, calculated as the discount value divided by the number of penalized units (*N*).

Algorithm 5 Equal Strategy

```

1: for all (discounted_label, penalized_list) in eligible_discounts do
2:   discounted_unit  $\leftarrow$  extract_logical_unit(discounted_label)
3:   discount_value  $\leftarrow$  discount_rule.discount_percentage
4:   results_by_logical_unit [discounted_unit]  $- =$  discount_value
5:   N  $\leftarrow$  |penalized_list|
6:   for all (plabel, value) in penalized_list do
7:     penalized_unit  $\leftarrow$  extract_logical_unit(plabel)
8:     results_by_logical_unit[penalized_unit]  $+=$  discount_value / N
9:   end for
10: end for

```

5.3.4.3 Reporting

The reporting module is responsible for generating a structured record of each allocation run. It operates as a post-processing stage of the cost allocation engine and executes immediately after the application of the discount mechanism. Each report is a markdown file, capturing all intermediary and final states of the allocation process for traceability and auditing.

Each report consists of five ordered sections. The first section, *Metric Results*, tabulates the normalized usage values computed per metric, segmented by relevant labels as specified in the configuration. Each table corresponds to a single metric and includes a mapping from label values to their normalized metric values.

The second section, *Allocation by Label*, presents the total cost assigned to each individual label after combining the metric weights and dimension weights. This allocation is directly derived from the normalized results of the previous section and constitutes the intermediate output prior to logical unit mapping.

The third section, *Allocation by Logical Unit (Before Discounts)*, displays the aggregated allocation results per logical unit, prior to the application of discount rules. This aggregation is computed by mapping each label to its corresponding logical unit using the configured extraction regex, with unmatched labels grouped under a reserved “others” category. This section explicitly traces the contributing labels for each logical unit, preserving the provenance of allocated values.

The fourth section, *Discounts*, enumerates all discount rules that were triggered in the run, along with their resulting impact. For each triggered discount, the report lists: the discounted label and its logical unit, and a table that specifies, for each penalty, the penalized label, the penalized logical unit, and the amount of value reallocated.

The final section, *Allocation by Logical Unit (After Discounts)*, records the final cost allocation after all discount rules have been applied. It reflects the adjusted values from the prior section and represents the output of the complete allocation pipeline.

Each report is stored in a designated output directory with a filename encoding the allocation timestamp. Reports may be independently validated for consistency and correctness, serving as ground truth for audit processes.

Chapter 6

Kafka Case Study

In this chapter, we examine a real-world application of the proposed cost allocation framework in a Kafka-based infrastructure. Apache Kafka, a distributed streaming platform, is selected as the subject of this case study due to its inherent multi-tenant architecture and complex resource utilization patterns. We describe the typical roles teams play in a shared Kafka deployment, detail how metrics used in the allocation model will be extracted, and outline the deployment characteristics that influence cost modeling. These components will be mapped to the configuration logic in the subsequent chapter.

6.1 Multi-Tenant Usage

Apache Kafka is a multi-tenant environment shared by different teams. Each team either produces, consumes, or governs Kafka topics within the centralized infrastructure; some teams perform only one of these roles, while others participate in all three simultaneously. This shared usage model introduces cost attribution challenges. Teams interact with Kafka in one or more of the following capacities:

- **Producers:** Teams publish messages to specific Kafka topics. These producers differ in message size, publishing frequency, and traffic patterns, all of which affect the load on Kafka brokers.
- **Consumers:** Teams subscribe to topics, forming consumer groups to process data streams. The number of consumers and their processing delay contribute to Kafka's operational cost and indirectly reflect the service's value.
- **Owners:** Some teams act as topic owners. Ownership implies functional responsibility for uptime, data schema governance, topic configuration, and coordination with platform teams.

This implicit ownership, combined with producer-consumer dynamics, results in a web of inter-dependencies where one team's topic may be consumed by multiple others, creating inter-team service relationships.

6.2 Metric Collection

As previously discussed, the cost attribution model relies on telemetry data exposed through Prometheus-compatible metrics. For Kafka, metric collection is implemented using two specialized Prometheus exporters: the JMX Exporter[22] and KMinion Exporter[23].

The JMX Exporter serves as the primary interface for exposing internal Kafka broker metrics. It leverages Java Management Extensions (JMX) to extract capture JMX MBean values.

In parallel, KMinion provides detailed observability to all valuable information that is accessible via the Kafka protocol.

The combination of these two exporters ensures visibility into Kafka's operational state.

6.3 Deployment Context

Before jumping into the cost allocation configuration for Kafka, we first need to define how Kafka is being deployed, since it highly affects the configuration, specifically the weights attributed to each dimension.

At Kuehne+Nagel, Kafka runs on top of Amazon Elastic Kubernetes Service (EKS) [30], which provides a managed Kubernetes control plane for orchestrating containerized workloads. Each Kafka broker is deployed as a Kubernetes pod, scheduled onto dedicated EC2 instances [9]. This approach ensures predictable resource provisioning and compatibility with the broader cloud-native ecosystem for monitoring and autoscaling.

The EC2 instances used in this deployment are of type t3.xlarge, offering 4 virtual CPUs and 16 GiB of memory.

Storage for each broker is provisioned using Amazon Elastic Block Storage (EBS) [2] with gp2 volumes, which offer SSD-backed performance with moderate baseline IOPS and burst capability. These volumes are mounted into the brokers to persist Kafka's log segments, ensuring durability and fast read/write access for active partitions.

The Kafka cluster is deployed within a single AWS Availability Zone (AZ) [24]. All Kafka brokers run in the same AZ, reducing inter-AZ data transfer costs. While this design does not provide the same level of fault tolerance as a multi-AZ deployment, it significantly reduces network overhead.

Kafka topics in this deployment are configured with a replication factor of three, meaning each partition is replicated across the three distinct brokers. This provides resilience against individual broker or instance failure.

Chapter 7

Evaluation and Validation

This chapter evaluates the proposed cost allocation framework using Kafka as target technology. Section 7.1 starts by defining fairness. Section 7.3 presents the experimental methodology used to extract costs. Section 7.4 details a simulation scenario based on metrics extracted from a production environment. Finally, Section 7.5 compares the framework with Conductor Chargeback Interceptor across multiple scenarios.

7.1 Definition of Fairness

Fairness is a highly subjective concept, making it difficult to establish a universal definition. Therefore, the definition of fairness must be derived from the company's core values. At Kuehne+Nagel, fairness is grounded in three key principles: Open Data, Good Governance Practices, and Accountable Consumption.

Fairness, in this environment, begins with the recognition that data is a shared asset whose value increases when it is openly accessible, responsibly governed, and efficiently consumed. Therefore, a fair system must reward teams for how they contribute to the data ecosystem.

From the Open Data perspective, fairness means that teams who make their Kafka topics openly available for consumption by other teams, and whose data is actively used, are recognized for their contribution. These teams enable cross-functional insights, reduce redundancy, and unlock efficiencies across the organization. A fair evaluation thus acknowledges this behavior and ensures that teams providing widely consumed, high-quality data are not penalized for the load that results from their generosity. Instead, they should be rewarded financially for enabling reuse and fostering collaboration.

Simultaneously, fairness incorporates the principle of Good Governance Practices. Teams that demonstrate strong governance practices, such as proper schema management, are actively reducing operational risk and improving Kafka's manageability. These teams ensure that shared infrastructure remains sustainable and reliable. Therefore, fairness requires that teams that uphold governance standards are not only recognized but also materially benefited for these efforts.

Finally, fairness must also be grounded in the principle of Accountable Consumption. Each team should be financially responsible for the Kafka resources they use. This usage-based cost attribution ensures transparency and prevents misuse or overconsumption, especially in a shared environment. No team should bear the burden of another's usage, and no team should be disincentivized from optimizing their own footprint.

Bringing these elements together, fairness in this Kafka-based, multi-team context means:

- Reward teams who contribute high-value, consumable data to others.
- Incentivize and acknowledge adherence to good governance practices.
- Accurately charges teams for their consumption, ensuring each pays their fair share.

This definition ensures that fairness is not merely about equal cost distribution, but rather about recognition of contribution, responsible behavior, and transparent consumption. It balances collective benefit with individual accountability and reflects the company's cultural emphasis on open data, responsibility, and collaboration.

In subsequent experiments, we evaluate whether the proposed allocation framework reflects these fairness principles by observing cost attribution outcomes under different simulated usage patterns. Specifically, we assess whether provider teams are financially rewarded, governance behavior is incentivized, and consumers are proportionally charged.

7.2 Simulator

To evaluate the behavior of the cost allocation framework under configurable load and topology scenarios, a custom simulator was developed.

The simulator operates from a JSON configuration file, which defines a list of participating teams. Each team block can specify a set of topics, producer definitions, and consumer groups associated with that team. Topics support additional configuration parameters beyond basic meta-data, including tuning settings such as the number of partitions and replication factor. Additionally, a preload size field (expressed in megabytes) can be included within each topic configuration. When present, this parameter triggers a preload operation, causing the simulator to inject data into the topic up to the specified size before live traffic begins. Producers can be configured by message size and message rate, allowing the simulation of diverse producing behaviors. Consumer groups can include any number of consumer instances and subscribe to specific topics.

During startup, the simulator reads this configuration, creates the required Kafka topics, preloads data, and initiates producer and consumer processes.

Message flow begins once all resources are initialized. Producers generate traffic toward their configured topics, while consumers process messages in parallel. The simulation runs indefinitely until externally interrupted.

This simulator provides a flexible and reproducible framework to simulate multi-tenant setups in Kafka.

7.3 Methodology for Cost Extraction

All experiments conducted in this chapter follow a standardized methodology designed to ensure consistency in cost extraction. The simulation lifecycle is structured in three distinct phases: infrastructure initialization, preload phase, and steady-state execution.

1. **Infrastructure Initialization:** Upon starting the simulation, the Kafka infrastructure is deployed and allowed to stabilize. This includes starting up the broker, initializing internal topics, and registering metrics exporters.
2. **Preload Phase:** Once the infrastructure is stable, the simulator performs an optional preload operation as defined in the scenario configuration. If specified, each topic is loaded with a predefined amount of data, simulating historical log size.
3. **Steady-State Execution:** After the preload is complete, producers and consumers are started. During this phase, after a short period of time, Kafka metrics rate stabilizes. Once metric values reach a consistent state, the cost allocation is extracted.

The rationale for this methodology is to avoid capturing transient behavior during startup or preloading and instead focus the analysis on a steady operating state. Only after metrics have reached stability are they considered for allocation purposes.

By adhering to this structured execution model across all experiments, the extracted allocation values are not only meaningful but also directly comparable between scenarios. This consistency is critical to evaluating cost allocation framework.

7.4 Real World Scenario

To evaluate the cost allocation, a simulation scenario was constructed based on service account (SA) metrics and consumption relationships provided by the organization. These metrics included, for each SA, the total storage size and the inbound average message rate over a 30-day window. Additionally, a directed graph of consumption relationships between service accounts was made available, indicating which SAs act as consumers of other SAs.

7.4.1 Simulator Setup

Given the number of topics involved in the context of the company, the simulation adopts a simplification strategy to maintain manageability. Each service account is represented by a single topic, abstracting away the underlying complexity of multiple real-world topics. This one topic per SA assumption preserves the semantics of inter-team communication while significantly reducing configuration overhead and the computational complexity associated with simulating and managing a high volume of topics, consumer groups, and producers. Given the scale of the original system, a full-fidelity simulation would demand substantial computing resources to replicate the complete topology.

In this setup, each SA is modeled as a producer that publishes messages to its corresponding topic at a rate equal to its inbound average message rate value. Message size is standardized across the simulation at 0.5 MB, which corresponds to the average message size observed in the production environment. For consumption, consumer groups are instantiated based on the provided SA dependency graph. If an SA is listed as consuming from another, a consumer group is created under the consuming SA, subscribing to the producing SA's topic.

Based on this data and these assumptions, a configuration was constructed to feed the simulator. This configuration holds the team's definitions, with topics, producers, and consumers associated with each SA.

For data protection and confidentiality reasons, all original service account identifiers were anonymized by converting them into an identifier.

This configuration ensures that the simulated Kafka traffic closely mirrors actual message flow patterns and service boundaries, while maintaining an analyzable and feasible environment. It captures both message ingress volume and inter-team data dependencies, allowing the cost allocation engine to be tested under realistic, multi-tenant conditions with meaningful producer-consumer relations.

7.4.2 Dataset Description

To characterize the behavior and structure of service interactions in the system, we analyze four core metrics derived from production data collected over a 30-day period from the organization's Kafka environment. The dependency relationships among Service Accounts (SA) were represented as a directed graph, where an edge from one SA to another indicates a consumption relationship, specifically, that the source SA consumes messages from the destination SA. In this formulation, the in-degree of a node represents the number of other SAs that consume from it, while the out-degree indicates the number of distinct SAs it consumes from.

The following metrics were used to define this scenario:

- **In-Degree:** The number of downstream SAs that consume messages from a given SA. This represents how many other services rely on the data produced by the SA.
- **Out-Degree:** The number of upstream SAs from which a given SA consumes messages. It indicates how many data sources the SA integrates or depends upon.
- **Message Rate:** The average number of messages ingested per second by each SA over the observed 30-day period.
- **Log Size:** The total amount of log data retained by each SA in megabytes.

A statistical analysis was conducted on the available dataset. The analysis was performed on a total of 387 SAs based on real-world metrics extracted from the organization's Kafka environment.

The in-degree distribution exhibited a notably high variance, with a mean of 1.89 and a standard deviation of 7.49. While the majority of nodes had low or zero in-degree, reflected in a median

of 0.00 and an interquartile range between 0.00 and 1.00, a small number of nodes exhibited disproportionately high in-degree values. The maximum observed in-degree was 94, indicating the presence of heavily consumed SAs acting as central data providers within the system. This suggests a highly skewed consumption pattern where a limited set of teams serve as critical data hubs. This is visually evident in both the boxplot shown in Figure 7.1 and the Top 10 most consumed SAs shown in Figure 7.5, where SA-104 alone is consumed by 94 others.

In contrast, the out-degree distribution was significantly more balanced. The mean remained at 1.89, yet with a much lower standard deviation of 1.65. The interquartile range extended from 1.00 to 2.00, and the maximum out-degree observed was 8.00. This comparatively narrow distribution indicates that most SAs consume from a small number of SAs. Unlike the in-degree distribution, there was no evidence of extreme outliers, suggesting that data consumption is more evenly distributed. This is clearly illustrated in the boxplot in Figure 7.2 and supported by the Top 10 consumers shown in Figure 7.6, where all values are tightly grouped between 6 and 8.

Regarding message ingress rates, the metric analyzed was the average number of messages per second (messages/sec) for each SA over a 30-day window. The distribution of this variable was also highly skewed. With a mean of 0.99 messages/sec and a standard deviation of 10.74, the dataset exhibits a significant long tail. The 25th, 50th (median), and 75th percentiles were all equal to 0.00, indicating that at least 75% of the SAs have negligible or no ingress activity. In contrast, the maximum observed message rate was 198.00 messages/sec, corresponding to a small set of SAs that generate substantial traffic. Figure 7.3 illustrates this skewed distribution through a boxplot, while Figure 7.7 highlights the dominance of SA-358 and a few others that produce high volumes of messages.

Similarly, the distribution of log sizes exhibits a highly skewed pattern. While the majority of SAs hold minimal data—evidenced by a 75th percentile of just 0.18 MB—a small number of accounts are responsible for the overwhelming majority of log data. The mean log size is 678.55 MB with a standard deviation of 6117.91 MB, and the maximum observed value reached 113,664 MB (approximately 113 GB). This is visually confirmed in Figure 7.4, where most SAs cluster near the bottom while several large outliers extend far beyond. The Top 10 log size contributors in Figure 7.8 show that SA-358 alone accounts for over 113 GB of logs—by far the most dominant contributor.

These findings highlight that the environment under study has a pronounced asymmetry in both the dependency structure, message ingress volume, and log size, with a small number of SAs accounting for the majority of consumption relationships and data production.

Table 7.1: Statistical Summary Scenario Metrics

Statistic	In-Degree	Out-Degree	Messages In Per Second (Avg 30d)	Log Size (MB)
Mean	1.89	1.89	0.99	678.55
Std Dev	7.49	1.65	10.74	6117.91
Min	0.00	0.00	0.00	0.00
25%	0.00	1.00	0.00	0.00
Median	0.00	1.00	0.00	0.00
75%	1.00	2.00	0.00	0.18
Max	94.00	8.00	198.00	113664.00

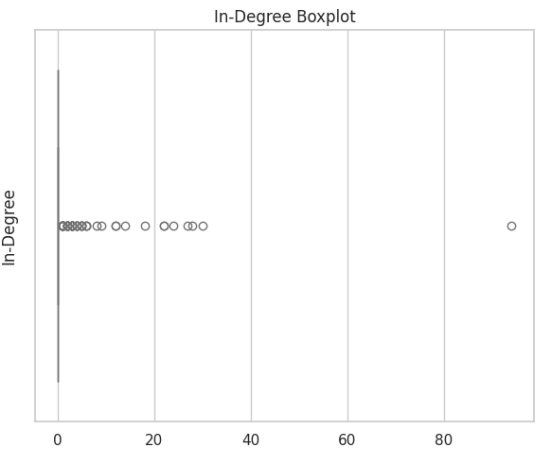


Figure 7.1: Boxplot of In-Degree Distribution

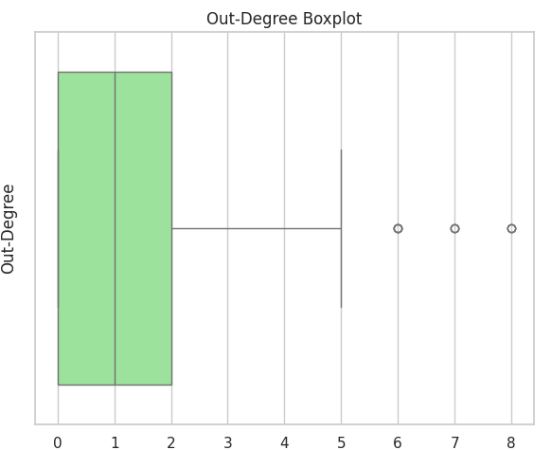


Figure 7.2: Boxplot of Out-Degree Distribution

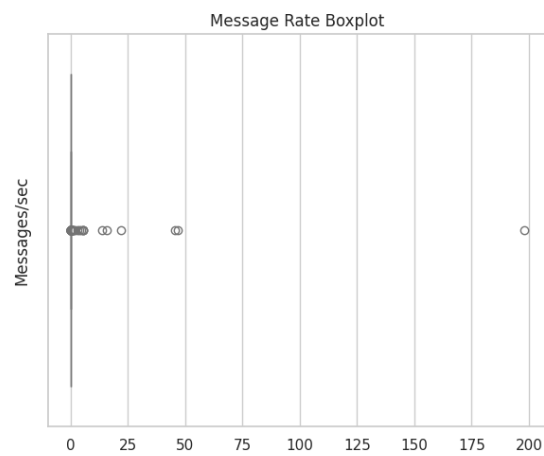


Figure 7.3: Boxplot of Message Ingress Rates

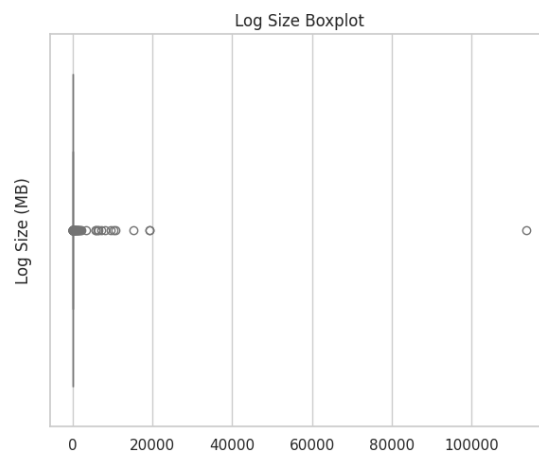


Figure 7.4: Boxplot of Log Sizes

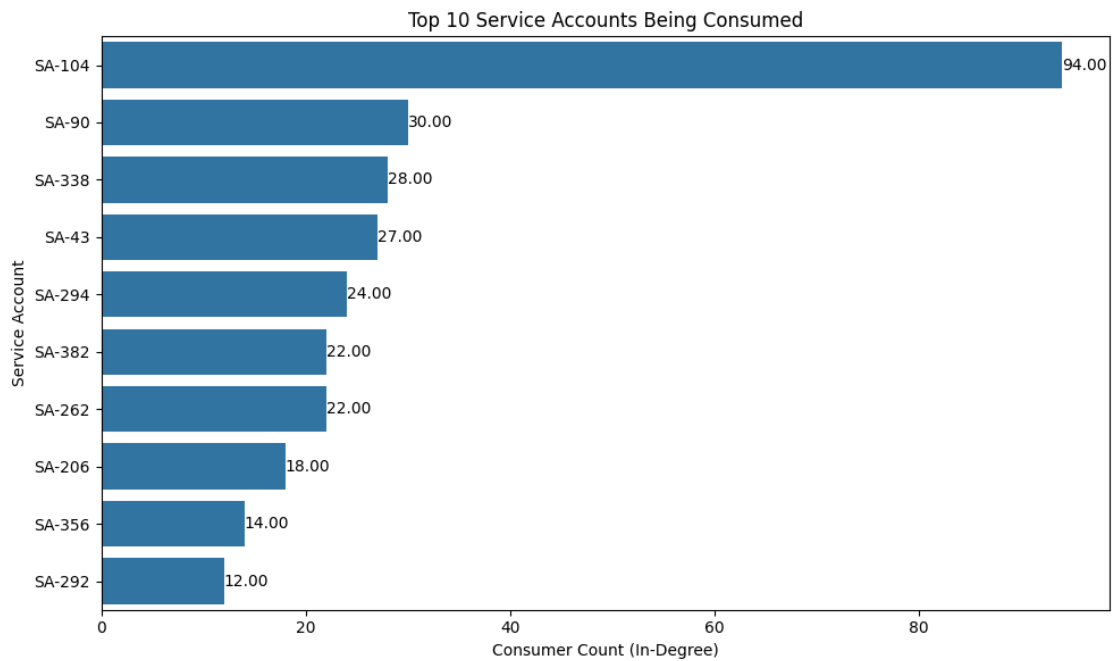


Figure 7.5: Top 10 Service Accounts by In-Degree (Being Consumed)

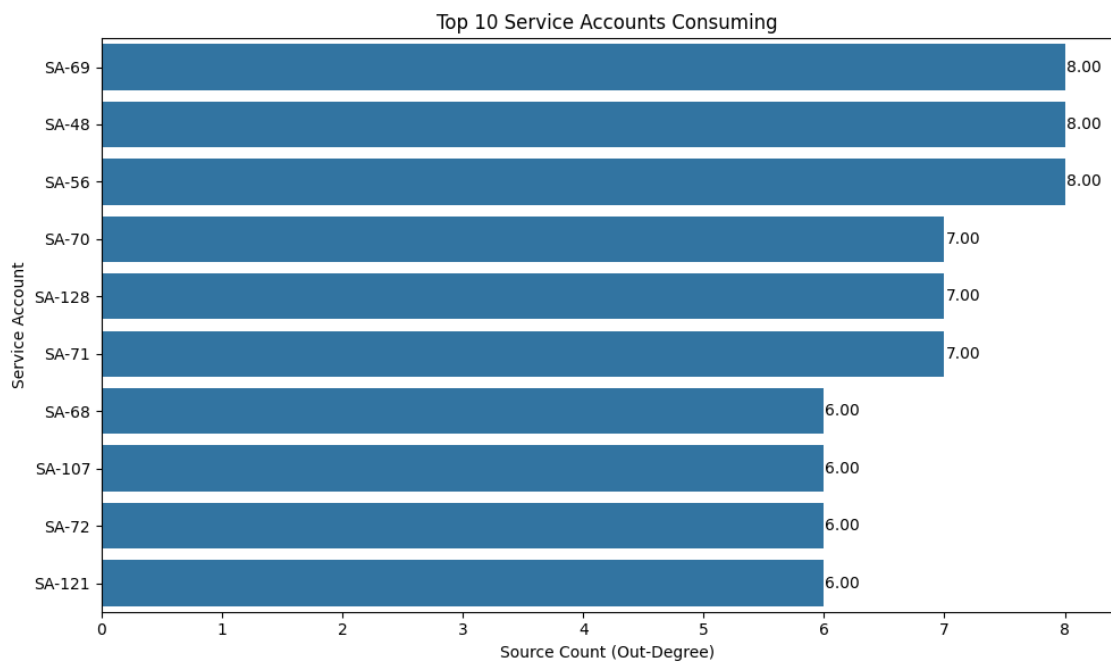


Figure 7.6: Top 10 Service Accounts by Out-Degree (Consuming)

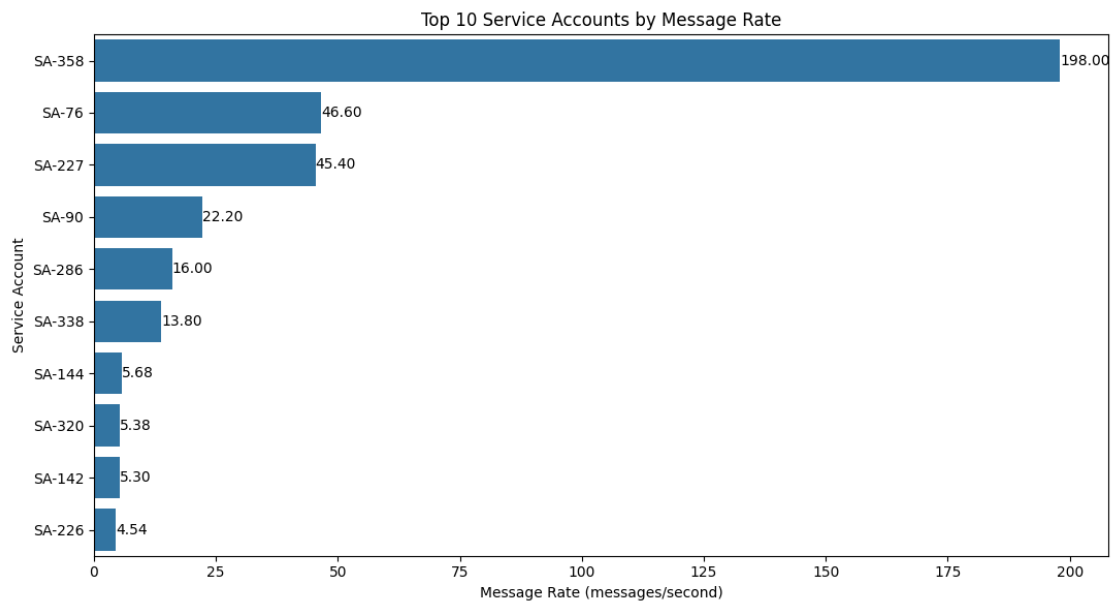


Figure 7.7: Top 10 Service Accounts by Message Rate

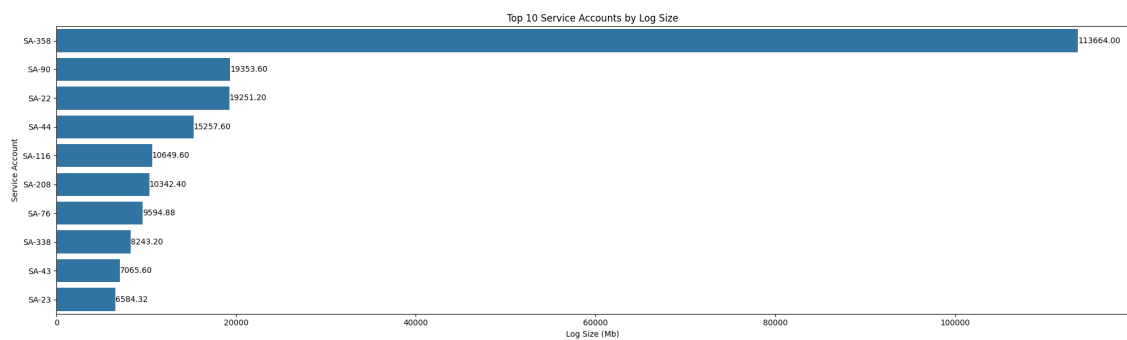


Figure 7.8: Top 10 Service Accounts by Log Size

7.4.3 Cost Allocation Configuration

This section introduces the configuration used for the cost allocation model regarding the real-world scenario, both in terms of dimensions and discounts.

7.4.3.1 Dimensions & Metrics

To construct the cost allocation configuration specific to Kafka, it is essential to identify the resource dimensions that most reflect infrastructure expenditure. As outlined in Confluent's blog post "A Guide to Mastering Kafka's Infrastructure Costs" [41], Kafka infrastructure costs typically decompose into three main categories: compute, storage, and network. These categories correspond to the CPU and memory resources required for broker operations, the disk space consumed by log segments, and the inter-broker and client data transfers across the network, respectively.

The cost impact of each dimension is highly dependent on Kafka's deployment topology. The Confluent blog emphasizes that in multi-AZ Kafka deployments, network costs can represent a substantial portion of the total infrastructure expense, primarily due to inter-broker replication traffic that crosses availability zone boundaries, which incurs AWS egress charges [32]. However, when brokers and clients are deployed entirely within a single Availability Zone, inter-node communication does not leave the zone, and therefore does not incur inter-AZ data transfer fees. In such cases, network traffic does not contribute to the infrastructure bill and can be excluded from the cost model. Consequently, the focus of the cost impact is restricted to the remaining two dimensions: storage and compute.

The Confluent blog also provides cost estimation formulas for each of the three resource types. In this configuration, only the equations for compute and storage are relevant, and their output will infer the weighting of each dimension in the allocation model. These are given as:

$$\begin{aligned} \text{Monthly storage cost} = & \frac{\text{Kafka MB ingress}}{\text{sec}} \times \frac{86,400 \text{ secs}}{\text{day}} \times \text{Retention days} \\ & \times \frac{0.001 \text{ GB}}{\text{MB}} \times \text{Replication factor} \times \frac{\text{Cost of GB}}{\text{month}} \text{ on EBS} \end{aligned} \quad (7.1)$$

$$\text{Monthly Compute Cost} = \text{Number of Nodes} \times \text{Hourly Cost per Node} \times 720 \quad (7.2)$$

As detailed in section 6.3, the deployment under analysis consists of 3 Kafka broker nodes, each running on a t3.xlarge EC2 instance, priced at \$0.192 per hour [33]. EBS storage is provisioned at \$0.119 per GB-month [27]. A retention period of 7 days is applied to all topics, and the replication factor is set to 3.

During the last month, at Kuehne+Nagel, Kafka processed 861,523,450 messages, with each message averaging 0.5 MB in size. The total monthly ingress can be calculated, using the above formulas, as:

$$861,523,450 \times 0.5 = 430,761,725 \text{ MB/month} \quad (7.3)$$

$$\frac{430,761,725}{30 \times 24 \times 3,600} = 166.17 \text{ MB/sec} \quad (7.4)$$

Using this ingress, the monthly storage cost is computed as:

$$166.17 \times 86,400 \times 7 \times 0.001 \times 3 \times 0.119 = \$3,047.46 \quad (7.5)$$

The compute cost, based on the number of nodes used, is:

$$3 \times 0.192 \times 720 = \$414.72 \quad (7.6)$$

These figures yield a total monthly Kafka infrastructure cost of \$3,462.18, where storage represents 88.0% and compute accounts for 12.0%. These percentages directly inform the dimensional weights used in the cost allocation configuration: 0.88 for storage and 0.12 for compute. This ensures that the cost distribution aligns closely with actual infrastructure usage and financial impact.

The following step to define the configuration is to identify the metrics associated with each dimension.

For the storage dimension, the metric used is *kafka_log_size*, collected and aggregated by topic. This metric captures the total amount of data stored on disk per topic, and therefore directly reflects the consumption of storage. It serves as an accurate reflection of per-topic storage usage.

For the compute dimension, it is difficult to map usage to topics in a precise, low-level way. Therefore, the approach must be more heuristic.

The metric used is *kafka_log_numlogsegments*, aggregated by topic. This metric reflects the number of log segments per topic. In Kafka, each log segment corresponds to a set of files on disk—including *.log*, *.index*, and *.timeindex* files, which implies the use of multiple file descriptors per segment. As noted in the blog "Deep dive into Apache Kafka storage internals: segments, rolling and retention" [14], "a partition is further split into segments, which are the actual files on the disk," and each partition directory contains not just the data file but also its associated index structures.

These files are actively managed by the broker. When segments are rolled—either because they reach a size limit, a time threshold, or because their index files become full—Kafka must close old files, flush buffers, and initialize new ones. This process introduces computational overhead related to file and index management. As described in the same blog, "a segment is rolled even when the corresponding index (or time index) is full," and each new segment triggers the creation and tracking of several files and their associated metadata.

Moreover, Kafka periodically runs background threads to check whether segments meet the conditions for deletion, as configured via retention policies. The more segments that exist, the more entries the broker must scan and evaluate during each cleanup interval. The blog explains that there is "an Apache Kafka thread that runs periodically on the broker to clean records and check which closed segments can be deleted," and this evaluation involves reading metadata across all closed segments. This mechanism means that the number of log segments has a direct impact on broker CPU utilization.

While *kafka_log_numlogsegments* is an indirect measure, it serves as a meaningful proxy for compute load. A higher number of segments leads to more open files, more index lookups, and more expensive background maintenance. All of which contribute to elevated compute resource usage on the broker.

The final configuration for the dimensions can be seen in listing 7.1.

```
1 "dimensions": {  
2   "storage": {
```

```

3      "weight": 0.88,
4      "metrics":[
5          {
6              "weight":1.0,
7              "name": "kafka_log_size",
8              "relevant_label": "topic"
9          }
10     ]
11 },
12 "compute":{
13     "weight": 0.12,
14     "metrics":[
15         {
16             "weight":1.0,
17             "name": "kafka_log_numlogsegments",
18             "relevant_label": "topic"
19         }
20     ]
21 }
22 }

```

Listing 7.1: JSON configuration Dimensions

7.4.3.2 Discounts

The final step in defining the cost allocation configuration is the introduction of discounts. In this configuration, four discount mechanisms are implemented.

The first discount (D1) shifts a portion of the allocated cost from the topic's owner to the consumer groups that read from that topic. It is activated by the metric *consumer_group_topic_offset_sum*, which captures the total number of offsets committed by each consumer group per topic. The topic's attributed cost is partially subtracted and proportionally redistributed among the consumer groups based on the number of committed offsets. The rationale is that consumers derive significant value from the data and contribute indirectly to system load, justifying a share of the topic's cost. Additionally, it is not fair for topic owners to bear the entire financial burden of a resource that is being consumed by others. A discount percentage of 0.1 (10%) was selected. Since the redistribution in this case is scoped specifically to the actual consumer groups associated with the topic, the dilution effect is limited to a relatively small subset of logical units. This makes it possible for even moderate discounts to result in perceptible cost shifts. The JSON configuration for this discount can be seen in listing 7.2.

```

1 {
2     "name": "high_topic_reader",
3     "trigger": "kminion_kafka_consumer_group_topic_offset_sum",
4     "discounted_label": "topic_name",
5     "penalized_label": "group_id",

```

```

6   "reallocation_strategy": "proportional",
7   "discount_percentage": 0.1
8 }

```

Listing 7.2: JSON configuration High Topic Reader Discount

The second discount (D2) promotes topics with a high byte egress-to-ingress ratio, as determined by the quotient of the metrics *bytes_out_per_sec_per_topic* and *bytes_in_per_sec_per_topic*. If this ratio exceeds a threshold of 2.0, the topic is interpreted as having high downstream relevance, meaning its data is actively consumed. In such cases, the topic owner receives a discount on the attributed cost, which is evenly redistributed across all other teams. Given that, such a strategy suffers from high dilution, redistributing small fractions of cost across a potentially large number of teams, a higher discount percentage is necessary to achieve a visible effect. Therefore, a discount percentage of 0.3 (30%) was selected. This mechanism incentivizes the design of topics that provide broad utility. The JSON configuration for this discount can be seen in listing 7.3.

```

1 {
2   "name": "valuable_topic_high_ratio_out_in",
3   "trigger": "kafka_server_total_bytes_out_per_sec_per_topic /
               kafka_server_total_bytes_in_per_sec_per_topic",
4   "discounted_label": "topic",
5   "discounted_condition": {
6     "compare_value": 2.0,
7     "operator": ">"
8   },
9   "penalized_label": "_all",
10  "reallocation_strategy": "equal",
11  "discount_percentage": 0.3
12 }

```

Listing 7.3: JSON configuration High Ratio Bytes Out/In Discount

7.4.4 Experimental Setup

The experiments were conducted in a containerized environment orchestrated with Docker Compose [15]. The Kafka infrastructure was deployed using the Confluent Platform [13] image with a single node acting as both broker and controller.

To support schema evolution, a Schema Registry [36] service was deployed and connected to the broker. For visualization and cluster management, Confluent Control Center was also included in the environment.

Monitoring infrastructure consisted of a Prometheus server scraping metrics from a Kafka Minion [23] and a JMX Exporter[22] connected to the broker. These metrics were made available for real-time visualization through a Grafana [25] instance, configured with data sources and dashboards.

The cost allocation engine was deployed as a dedicated container. It periodically scrapes Prometheus for metric data and exposes the computed cost allocations back as Prometheus-compatible metrics for observability.

The simulator responsible for generating Kafka traffic was executed externally, outside the Docker environment. It connected to the Kafka broker and was configured per scenario using a declarative JSON specification.

7.4.5 Results

This section presents a detailed analysis of the simulation results and is structured into three parts. First, it provides a descriptive and statistical overview of the allocation outputs before and after the application of discount mechanisms. Second, it explores the correlation between scenario input metrics and the resulting cost allocations. Finally, it presents a discussion of the findings in relation to the previously established definition of fairness.

7.4.5.1 Analysis of Allocation Results

Table 7.2 summarizes the key statistical metrics of the allocation values assigned to logical units, both before and after the application of the discount mechanism. Notably, the mean cost allocation remains constant at approximately 0.002577, which confirms that the discount mechanism is a redistributive process and does not alter the total allocation. However, the spread of the data changes significantly.

Before the application of discounts, the standard deviation was approximately 0.02198, indicating a relatively narrow distribution of costs. After discounts are applied, the standard deviation rises to 0.07842, reflecting a substantial increase in dispersion. This shift implies that the redistribution effect has meaningfully differentiated cost allocations across logical units. Moreover, the minimum value drops from 0 to -0.392, indicating that some logical units were recipients of discounts higher than their original costs.

The interquartile range also expands, shifting from a tightly bound interval around 0.000155 ($Q1=0.000155$, $Q3=0.000157$) before the discount to a broader spread ($Q1=0.008772$, $Q3=0.017571$) after the discount. These results reinforce that while the base allocation produced homogeneous cost values, the discount mechanism introduced variability, resulting in more differentiated cost attributions.

Table 7.2: Statistical Summary Results

Statistic	Base Allocation Value	After Discounts Value
Mean	0.002577	0.002577
Std Dev	0.021978	0.078420
Min	0.000000	-0.392010
25%	0.000155	0.008772
Median	0.000155	0.008774
75%	0.000157	0.017571
Max	0.413530	0.425480

7.4.5.2 Discount Distribution

Two discount rules were applied in the scenario as detailed in section 7.4.3.2. The first discount targeted teams whose topics were consumed by a number of distinct consumers, while the second rewarded teams whose topics generated a high ratio of bytes out and bytes in. In total, 58 logical units qualified under the "High Topic Reader" discount, while 11 qualified under the "Valuable Topic with High Out-In Ratio" discount. Importantly, all 11 teams that qualified under the latter also qualified under the former, indicating an overlap of particularly high-value producers.

7.4.5.3 Results Correlation with Scenario Metrics

Having analyzed the raw allocation values before and after applying the discounts, we next explore how scenario-level input metrics correlate with the computed costs.

A Spearman correlation analysis was conducted between scenario input variables and the allocated values at each phase of the process.

The Spearman rank correlation coefficient [40], denoted by ρ , is a non-parametric measure of statistical dependence between two variables. It assesses how well the relationship between two variables can be described by a monotonic function, without requiring the relationship to be linear or the data to be normally distributed. Spearman correlation is based on the ranked values of the variables rather than their raw values, making it robust to outliers and appropriate for ordinal data or data with non-linear relationships.

The results show strong positive correlations between the base allocation and the message rate ($\rho = 0.9577$) and log size ($\rho = 0.9169$). This is consistent with the design of the allocation configuration, which integrates directly the log size into the base cost allocation and indirectly, the message rate contributes for the increasing of the log size. Additionally, the results show a moderate correlation between the base allocation and the number of consumers ($\rho = 0.5189$). Since the number of consumers is not directly used in the base allocation, this is probably due to an indirect relationship, as message rate and number of consumers are themselves moderately correlated ($\rho = 0.5509$). Weaker correlations with the number of consuming relationships ($\rho = 0.1164$) suggest that this metric doesn't have a direct impact on the base allocation.

In contrast, correlations with the final allocation values after discounts are more nuanced. Notably, the number of consumers exhibits a negative correlation with the final cost ($\rho = -0.5465$), indicating that high-consumption producers benefited from discounts. Conversely, message rate and log size show slightly negative or negligible correlations with post-discount costs ($\rho = -0.1234$ and $\rho = -0.1370$, respectively), suggesting that these metrics, though primary contributors to the base allocation, are not related to the redistribution.

Furthermore, correlations between the discount applied and scenario metrics reveal some meaningful patterns: a moderate positive correlation with the number of consumers ($\rho = 0.5448$) and a strong negative correlation with the number of consuming relationships ($\rho = -0.6637$). This confirms that discounts tend to benefit teams with many consumers and penalize those consuming from a wide range of topics.

Table 7.3: Spearman Correlation Coefficients between Scenario Metrics and Allocation Variables

Variable A	Variable B	Spearman Correlation
Message Rate	Base Allocation Value	0.9577
Log Size	Base Allocation Value	0.9169
Number of Consumers	Base Allocation Value	0.5189
Number of Consuming	Base Allocation Value	0.1164
Number of Consumers	Discount	0.5448
Number of Consuming	Discount	-0.6637
Message Rate	Discount	0.2615
Log Size	Discount	0.2770
Before Discount Value	Discount	0.2678
Number of Consumers	After Discount Value	-0.5465
Message Rate	After Discount Value	-0.1234
Log Size	After Discount Value	-0.1370
Message Rate	Number of Consuming	0.1289
Message Rate	Number of Consumers	0.5509

7.4.6 Discussion

The results obtained from the simulation and statistical analysis can be interpreted in light of the fairness definition established earlier in section 7.1. The fairness model is founded on three guiding principles: Accountable Consumption, Open Data, and Good Governance Practices. The governance aspects were not explicitly demonstrated in the simulation scenario, but will be in a later scenario.

Accountable Consumption requires that teams are charged in proportion to their usage of shared infrastructure resources. The observed strong Spearman correlations between the base allocation and the two main usage metrics, average message rate ($\rho = 0.9577$) and log size ($\rho = 0.9169$), demonstrate that the initial allocation phase accurately reflects raw consumption

behavior. Although the network is not introduced in this configuration, it is highly correlated with the increase in log size. This demonstrates that teams with higher Kafka throughput or storage use are proportionally charged.

Moreover, the relatively low correlation between the number of consuming relationships and base allocation ($\rho = 0.1164$) confirms that the base allocation is agnostic to topological relationships and purely usage-driven. This isolation from topology affirms that only direct resource use affects cost, which is the intended behavior under the Accountable Consumption principle.

Open Data reiterates that teams that produce widely consumed data should be rewarded. The discount phase of the allocation mechanism directly targets this aspect of fairness, by rewarding the producers and penalizing the consumers

The analysis shows a moderate positive correlation between the number of consumers and the total discount received ($\rho = 0.5448$), suggesting that teams with a high number of consumers are regularly recognized via reduced financial burden. Similarly, the number of consuming relationships shows a strong negative correlation with discounts ($\rho = -0.6637$), meaning teams that act as broad consumers of data are financially penalized. These patterns confirm that the discount mechanism promotes data producers that have many consumers, which is aligned with the objective of rewarding the contribution of open data.

The inverse correlation between the number of consumers and final allocated cost ($\rho = -0.5465$) reinforces that the redistribution mechanism is successfully transferring costs away from central producers to broader consumers, as prescribed by the Open Data concept. These observations support the effectiveness of the discount model in promoting collaborative and transparent data sharing.

Despite the generally positive alignment between the allocation model and fairness principles, a few unexpected behaviors were observed. For instance, after discounts are applied, core usage metrics such as message rate and log size exhibit weak negative correlations with the final cost ($\rho = \sim 0.1234$ and $\rho = \sim 0.1370$, respectively). This suggests a possible overcompensation during the discount phase, where high-volume teams may end up with disproportionately low final costs. While this supports incentivizing data producers, it may undermine the principle of accountable consumption. This tension reveals a deeper contradiction within the fairness definition itself. While the Open Data principle justifies rewarding teams that produce widely consumed data, this reward can effectively reduce their final costs below what their raw usage would suggest. As such, high-volume producers might pay less than smaller consumers, contradicting the Accountable Consumption principle. In other words, incentivizing openness may inadvertently penalize lean consumers, who end up getting a greater share of the infrastructure cost. This trade-off exposes a fairness paradox: one principle prioritizes contribution and utility, while the other prioritizes consumption accountability.

Overall, the two-phase allocation model demonstrates clear alignment with the conceptual framework of fairness adopted in this work. The base allocation enforces individual accountability, while the discount phase rebalances cost in a way that acknowledges cross-team collaboration and data value creation. These findings confirm that the framework not only captures usage accurately

but also aligns cost incentives with organizational values of collaboration, accountability, and data openness.

7.5 Comparative Evaluation

This section presents a comparative evaluation of the proposed cost allocation framework against the Conductor Chargeback Interceptor [8]. The goal is to assess how each system performs under a variety of scenarios, with a particular focus on the impact of the multi-dimensional weighting and the discount mechanism. For each scenario, both the Conductor Chargeback and the proposed framework are executed under the same conditions, using the same configuration for the simulator. The comparison aims to highlight the behavioral differences between the two systems with respect to cost allocation and to evaluate whether the proposed framework yields allocations that better reflect usage and fairness.

In all scenarios, the cost allocation is computed independently by each framework. For scenarios that are designed purely to evaluate proportional resource attribution, such as single or multiple producers without consumers, discount mechanisms are intentionally disabled. For more complex scenarios that involve data reuse, schema governance, or consumer performance characteristics, selective discount mechanisms are applied in the proposed framework in order to specifically evaluate them. These are configured based on the nature of the scenario.

7.5.1 Experimental Setup

The experimental environment used to evaluate the proposed framework remains consistent with the setup described in the Real-World Scenario.

For the Conductor Chargeback Interceptor [8], a separate containerized environment was configured using Docker Compose [15]. This setup included the Conductor Console [12] and the Conductor Gateway [31], which acts as a proxy between clients and the Kafka broker. All Kafka traffic, including producer and consumer requests, was routed through the gateway.

The Conductor Gateway was configured with an observability interceptor, which captures usage metrics needed for cost allocation. These metrics are processed internally by Conductor to compute service-account-based chargebacks. Service accounts were created via the Conductor Gateway's REST API [6], and all Kafka operations were performed using SASL authentication [39] with the appropriate service account credentials.

The Kafka broker in the Conductor setup was deployed with support for SASL/PLAIN authentication. The schema registry was also deployed to support experiments requiring schema evolution and governance.

The simulator was reused in this setup but connected through the Conductor Gateway, thereby ensuring that all traffic flowed through the interceptor, enabling metric collection by the Conductor console.

7.5.2 Cost Allocation Configuration

This section introduces the configuration used for the cost allocation model regarding the comparative analysis scenarios, both in terms of dimensions and discounts.

7.5.2.1 Dimensions & Metrics

As done before in the real world scenario, the dimensions and respective weights were inferred using Confluent’s blog post “A Guide to Mastering Kafka’s Infrastructure Costs” [41]. In the last scenario, we did not consider the network dimension since our deployment context was only using a single availability zone. Since the Conductor Chargeback Interceptor only considers bytes In and bytes Out per service account, we decided to, in order to add the network dimension, consider three availability zones. Therefore, to complete the dimensional cost model, we must now derive the relative weight of the network dimension compared to storage and compute. The computation and storage costs have already been established in the previous scenario as \$414.72 and \$3,047.46 per month for an ingress rate of 166.17 MB/s. These figures were based on observed message volume, instance pricing, retention duration, and the replication factor, as detailed in Section 7.4.3.1.

In the new scenario, the transition to a three-Availability Zone Kafka deployment introduces cross-AZ data transfer charges as a dominant cost component. Following Confluent’s model, cross-AZ throughput includes traffic from producers writing to remote partition leaders, consumers reading from leaders in other zones, and replication traffic between AZs. The total cross-AZ data transfer is estimated as:

$$\text{Cross-AZ Throughput (MB/sec)} = \frac{2}{3} \times \text{Ingress} + \frac{2}{3} \times \text{Egress} + 2 \times \text{Ingress} \quad (7.7)$$

Assuming ingress and egress are both the same, 166.17 MB/sec, the resulting cross-AZ throughput is approximately 553.90 MB/sec. Using the Cross-Az Throughput, the monthly network cost is given by the equation:

$$\begin{aligned} \text{Monthly Networking Cost} = & \text{Cross-AZ Throughput (MB/sec)} \times 2,592,000 \times 0.001 \\ & \times \text{Cost per GB Cross-AZ} \end{aligned} \quad (7.8)$$

Over a month, with AWS pricing for inter-AZ traffic at \$0.01 per GB, the monthly network cost is:

$$553.90 \times 2,592,000 \times 0.001 \times 0.01 = \$14,357.088 \quad (7.9)$$

Summing across all three dimensions yields a total monthly Kafka infrastructure cost of \$17,819.26. This cost composition informs the dimensional weights as follows:

$$\text{Storage Weight} = \frac{3,047.46}{17,819.26} \approx 0.17 \quad (7.10)$$

$$\text{Compute Weight} = \frac{414.72}{17,819.26} \approx 0.02 \quad (7.11)$$

$$\text{Network Weight} = \frac{14,357.08}{17,819.26} \approx 0.81 \quad (7.12)$$

The resulting cost allocation configuration introduces the network dimension explicitly. The metrics chosen for the network dimension are *kafka_server_total_bytes_out_per_sec_per_topic* and *kafka_server_total_bytes_in_per_sec_per_topic*, capturing both ingress and egress from topics.

The final configuration is shown in Listing 7.4:

```

1 "dimensions": {
2     "storage":{
3         "weight": 0.17,
4         "metrics":[
5             {
6                 "weight":1.0,
7                 "name": "kafka_log_size",
8                 "relevant_label": "topic"
9             }
10        ]
11    },
12    "network": {
13        "weight": 0.81,
14        "metrics":[
15            {
16                "weight": 0.5,
17                "name": "kafka_server_total_bytes_out_per_sec_per_topic",
18                "relevant_label": "topic"
19            },
20            {
21                "weight": 0.5,
22                "name": "kafka_server_total_bytes_in_per_sec_per_topic",
23                "relevant_label": "topic"
24            }
25        ]
26    },
27    "compute":{
28        "weight": 0.02,
29        "metrics":[
30            {
31                "weight":1.0,
32                "name": "kafka_log_numlogsegments",
33                "relevant_label": "topic"

```

```

34         }
35     ]
36 }

```

Listing 7.4: JSON configuration Dimensions for Multi-AZ Deployment

7.5.2.2 Discounts

Two discounts were proposed in the real-world scenario: "High Topic Reader" (D1) and "High Ratio Bytes Out/In" (D2). For the following scenarios, two more discounts were created.

The first discount (D3) rewards consumer groups that consistently maintain low lag. The metric *kminion_kafka_consumer_group_topic_lag* captures the number of unprocessed messages for a given consumer group per topic. When the observed lag for a consumer group remains below a threshold of 50 messages, it is interpreted as evidence of responsive and efficient consumption behavior. In such cases, the corresponding consumer group is granted a discount on its allocated cost. The deducted cost is redistributed equally across all other teams. While the redistribution strategy remains global and equal, the nature of the metric condition is such that a significant number of consumer groups may qualify simultaneously, particularly in environments with low-latency requirements and well-optimized consumption pipelines. As a result, the discount percentage may be moderate. A discount percentage of 0.1 (10%) provides a reasonable incentive for sustained consumer responsiveness without over-discounting. This mechanism incentivizes timely data processing and encourages teams to optimize their consumer implementations to avoid lag accumulation, which can otherwise degrade overall system performance. The JSON configuration for this discount can be seen in listing 7.5.

```

1 {
2     "name": "valuable_consumer_group_low_consumer_lag",
3     "trigger": "kminion_kafka_consumer_group_topic_lag",
4     "discounted_label": "group_id",
5     "discounted_condition": {
6         "compare_value": 50,
7         "operator": "<"
8     },
9     "penalized_label": "_all",
10    "reallocation_strategy": "equal",
11    "discount_percentage": 0.1
12 }

```

Listing 7.5: JSON configuration Low Consumer Lag

The fourth discount (D4) targets teams that enforce strict schema compatibility rules on their topics. It is triggered by the presence of the *compatibility_type* attribute retrieved from the Schema Registry [36]. Specifically, the discount is applied to topics whose compatibility mode is not

NONE, indicating the enforcement of compatibility guarantees across schema versions. When this condition is met, the topic owner receives a cost reduction, and the discounted amount is evenly distributed across all teams. Since this discount is redistributed globally, its effectiveness depends on the relative rarity of the qualifying behavior. In organizations where schema governance is inconsistently applied or only partially enforced, a discount of 0.1 (10%) is justified to make the effect visible and to actively encourage broader adoption. A lower value may fail to differentiate compliant from non-compliant teams, while a higher value could distort cost allocation for relatively small infrastructure benefits. This policy rewards strong governance practices that improve data contract stability and reduce integration risk. By promoting adherence to schema governance, the discount contributes to the long-term maintainability and robustness of the data ecosystem. The JSON configuration for this discount can be seen in listing 7.6.

```
1 {
2   "name": "schema_governance",
3   "trigger": "compatibility_type",
4   "discounted_label": "topic",
5   "discounted_condition": {
6     "compare_value": "NONE",
7     "operator": "!="
8   },
9   "penalized_label": "_all",
10  "reallocation_strategy": "equal",
11  "discount_percentage": 0.1
12 }
```

Listing 7.6: JSON configuration Schema Compatibility

7.5.3 Discounts Configuration Per Scenario

These discounts are not used uniformly across all scenarios. In the initial scenarios, discounts are intentionally disabled. This allows a focused evaluation of the base allocation behavior under multi-dimensional weighting. By removing discounts redistribution, these scenarios isolate how each framework translates raw usage metrics into cost attribution and whether a more expressive, weighted dimensional model leads to a fairer outcome.

Discounts are only introduced in later scenarios, where the goal is to examine higher-level fairness dynamics such as collaboration and governance. Each discount is selectively activated based on the structural and behavioral attributes of the scenario in question.

Table 7.4: Summary of Discount Application per Scenario

Scenario	D1	D2	D3	D4
Single Producer (S.01)	X	X	X	X
Multiple Producers Same Rate (S.02)	X	X	X	X
Multiple Producers Same Rate with Different Initial Storage (S.03)	X	X	X	X
Multiple Producers Different Rate (S.04)	X	X	X	X
One Producer, Multiple Consumers (S.05)	✓	✓	X	X
Schema Governance (S.06)	X	X	X	✓
Multiple Producers Same Topic (S.07)	X	X	X	X
Consumer Lag (S.08)	✓	X	✓	X
Different Topics, One Consumer (S.09)	✓	X	X	X

7.5.4 Experiments

The following subsections detail the experimental scenarios used to evaluate both systems. Each scenario is designed to isolate a specific behavior or fairness objective, and the results from Conduktor and the proposed framework are compared to highlight their relative effectiveness.

7.5.4.1 Experimental Considerations

As previously mentioned, in the proposed framework, a small portion of the cost is allocated to an artificial logical unit labeled "others". This unit accounts for labels that cannot be matched to any logical unit using the defined regular expression, typically originating from internal topics. Since Conduktor does not address this type of label, the percentage assigned to "others" in the proposed framework will be evenly redistributed across all other logical units. This cost is consistently minimal and does not significantly affect the overall results. The meaning of each symbol used in the scenario diagrams is provided in image 7.9.

	Producer
	Consumer
	Topic
	Produce to at X Msg/s
	Consume From

Figure 7.9: Caption for the symbols used in the scenarios diagram

7.5.4.2 Scenario 1 – Single Producer (S.01)

This scenario defines the baseline case, where a single service account (SA-1) has a producer, publishing to its own dedicated topic. There are no consumers or additional services involved. The objective is to verify whether both cost attribution systems correctly allocate the entire resource consumption cost to the sole active participant, in the absence of inter-service interactions or performance variability. This baseline scenario is designed to evaluate base allocation behavior without any redistribution logic. Therefore, all discounts are explicitly disabled to isolate direct cost attribution.

The results show that both systems allocated 100% of the cost to SA-1.

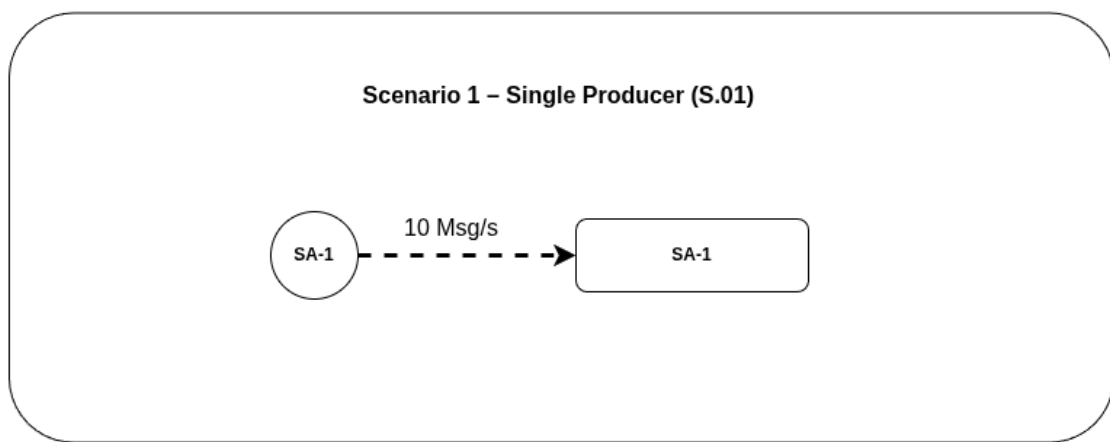


Figure 7.10: Scenario S.01 Diagram

Table 7.5: Scenario S.01 Allocation Results

System	Logical Unit	Base Allocation
Conduktor	SA-1	1.00
Proposed Framework	SA-1	1.00

7.5.4.3 Scenario 2 – Multiple Producers Same Rate (S.02)

This scenario involves two independent producers, SA-1 and SA-2, each writing to their own distinct topics at the same message rate and message size. No consumers are present. The purpose is to evaluate whether the cost attribution systems maintain symmetry and fairness by evenly distributing costs across multiple producers with equivalent workload characteristics. Like Scenario S.01, this scenario is designed to evaluate base allocation behavior without any redistribution logic. Therefore, all discounts are deactivated.

Both allocation systems produced symmetric results, assigning 50% of the total cost to each producer.

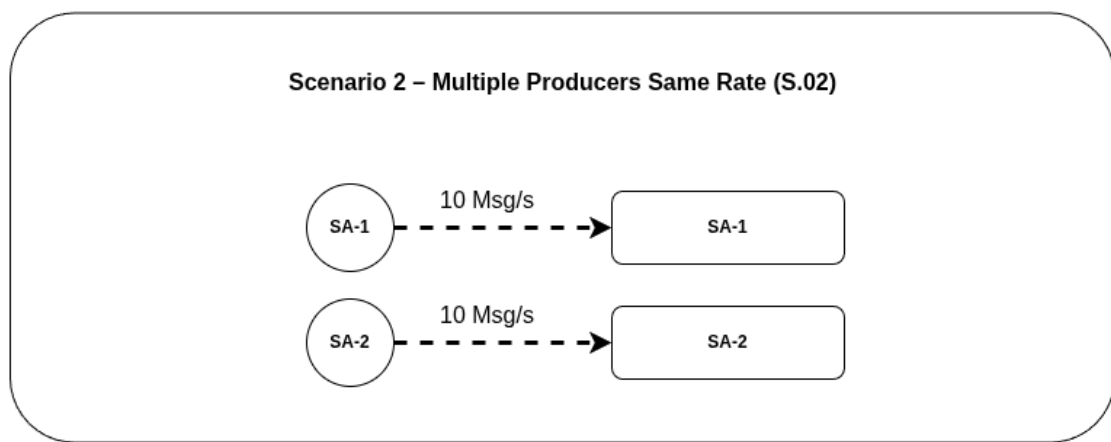


Figure 7.11: Scenario S.02 Diagram

Table 7.6: Scenario S.02 Allocation Results

System	Logical Unit	Base Allocation
Conduktor	SA-1	0.50
Conduktor	SA-2	0.50
Proposed Framework	SA-1	0.50
Proposed Framework	SA-2	0.50

7.5.4.4 Scenario 3 – Multiple Producers Same Rate with Different Initial Storage (S.03)

Building upon the configuration of Scenario 2, this scenario introduces an asymmetry in historical topic storage. SA-1 and SA-2 both produce to separate topics at the same rate. However, SA-1's topic is preloaded with 500 MB of retained data prior to the experiment, whereas SA-2's topic starts with empty storage. The objective of this scenario is to assess how each cost allocation system accounts for multiple variables, storage, and network. Therefore, discounts are not applied to access the multi-dimensional weighting of the proposed framework.

Conduktor attributed equal cost to both service accounts, assigning 0.50 to SA-1 and 0.50 to SA-2. In contrast, the proposed framework allocated a higher share of the cost to SA-1 (0.56) and a lower share to SA-2 (0.44), reflecting the retained data size difference.

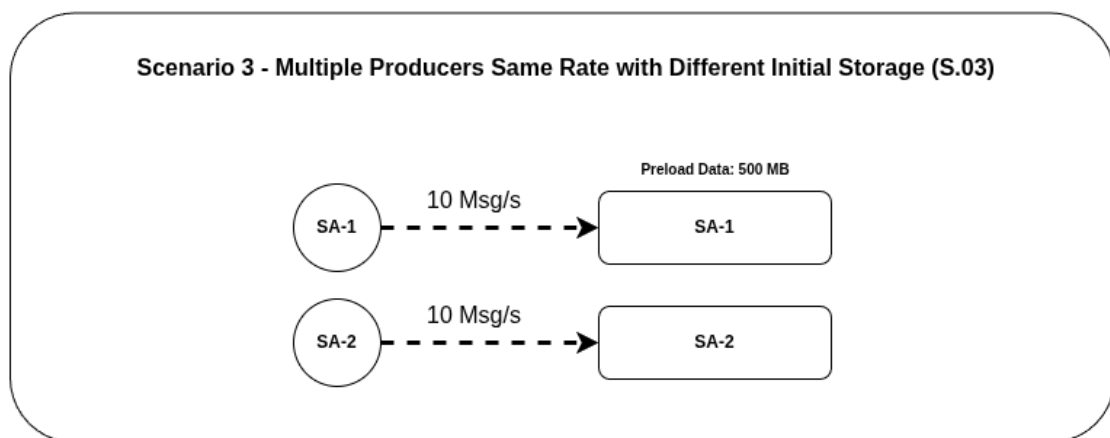


Figure 7.12: Scenario S.03 Diagram

Table 7.7: Scenario S.03 Allocation Results

System	Logical Unit	Base Allocation
Conduktor	SA-1	0.50
Conduktor	SA-2	0.50
Proposed Framework	SA-1	0.56
Proposed Framework	SA-2	0.44

7.5.4.5 Scenario 4 – Multiple Producers Different Rates (S.04)

In this scenario, SA-1 and SA-2 both publish to their own topics, but at unequal message rates: SA-1 transmits at 20 messages per second, and SA-2 at 10 messages per second. There are no consumers. The goal is to test whether the allocation systems proportionally assign resource costs based on relative throughput, thus validating their sensitivity to dynamic production volumes. This scenario tests responsiveness to message rate disparities. Discounts are withheld to isolate the allocator's sensitivity to throughput asymmetry.

Conduktor reported a higher allocation to SA-1 (0.6662) and a lower value to SA-2 (0.3368), approximating a 2:1 ratio. The proposed framework produced more balanced values: 0.595 for SA-1 and 0.405 for SA-2

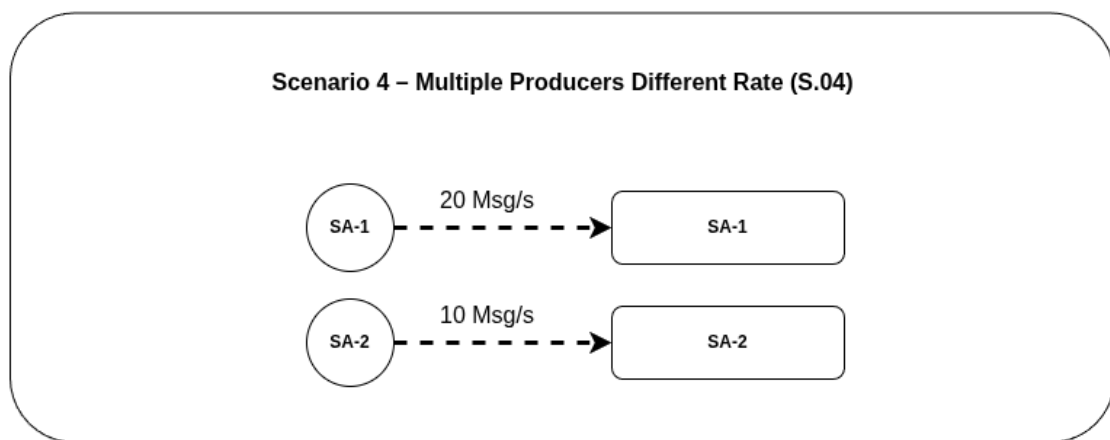


Figure 7.13: Scenario S.04 Diagram

Table 7.8: Scenario S.04 Allocation Results

System	Logical Unit	Base Allocation
Conduktor	SA-1	0.6662
Conduktor	SA-2	0.3368
Proposed Framework	SA-1	0.595
Proposed Framework	SA-2	0.405

7.5.4.6 Scenario 5 – One Producer, Multiple Consumers (S.05)

This scenario introduces SA-1 as a producer and SA-2 and SA-3 as consumers of SA-1's topic. The aim is to verify whether the proposed framework can redistribute cost responsibility away from the producer and toward the downstream consumers, reflecting their contribution to resource usage. In this scenario, the first two discounts are used. The first one shifts costs from topic owners to consumers, and the second one rewards topics with a high bytes out/in ratio.

Conduktor distributed the cost evenly across all three agents, each receiving an allocation of 0.33. In contrast, the proposed framework initially attributed 1.0 to SA-1, which was reduced to 0.58 after discounts, while SA-2 and SA-3 each received a final allocation value of 0.21.

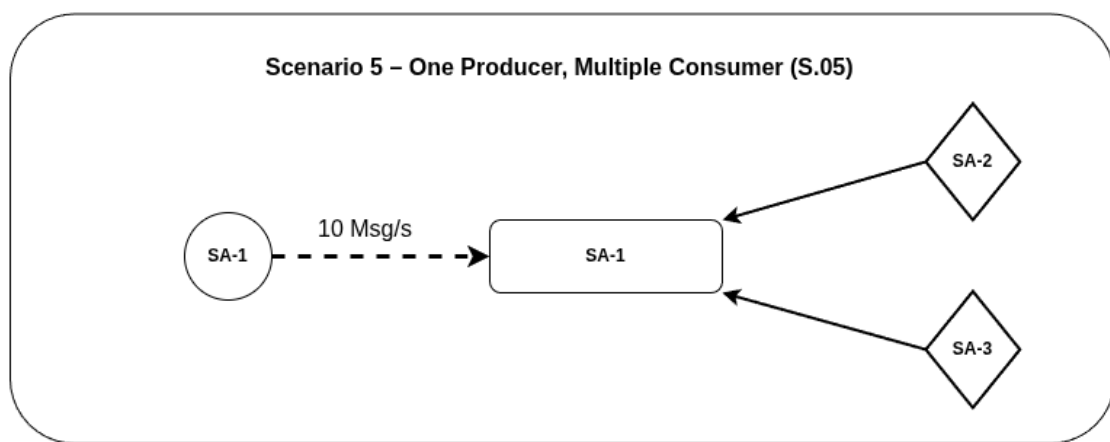


Figure 7.14: Scenario S.05 Diagram

Table 7.9: Scenario S.05 Allocation Results

System	Logical Unit	Base Allocation	After Discounts
Conduktor	SA-1	N/A	0.33
Conduktor	SA-2	N/A	0.33
Conduktor	SA-3	N/A	0.33
Proposed Framework	SA-1	1.0	0.58
Proposed Framework	SA-2	–	0.21
Proposed Framework	SA-3	–	0.21

7.5.4.7 Scenario 6 – Schema Governance (S.06)

In this scenario, two producers, SA-1 and SA-2, publish to their own topics. The topic associated with SA-1 has schema compatibility disabled (`NONE`), while SA-2's topic enforces `FULL_TRANSITIVE` compatibility. Schema compatibility types were configured via the schema register, and a specialized scraper in the proposed framework was used to extract these values. This scenario isolates the effect of schema governance on cost attribution by applying only the schema-based discount. The objective is to assess whether the system can reward governance practices that improve data interoperability and quality assurance. Only the schema governance discount (D4) is enabled to reward schema enforcement practices.

Conduktor assigned an equal cost of 0.50 to both service accounts. The proposed framework began with equal values (0.50 each), but the application of discounts resulted in SA-1 receiving 0.60 and SA-2 receiving 0.40.

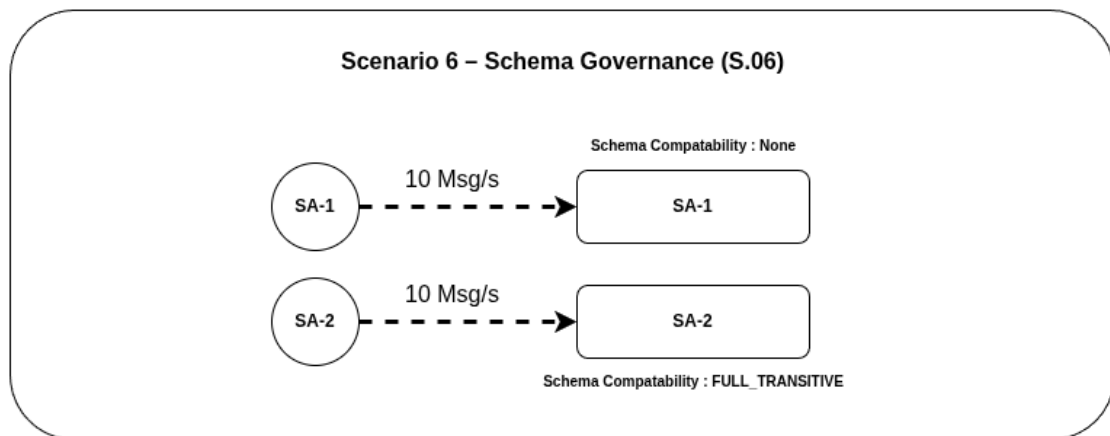


Figure 7.15: Scenario S.06 Diagram

Table 7.10: Scenario S.06 Allocation Results

System	Logical Unit	Base Allocation	After Discounts
Conduktor	SA-1	N/A	0.50
Conduktor	SA-2	N/A	0.50
Proposed Framework	SA-1	0.50	0.60
Proposed Framework	SA-2	0.50	0.40

7.5.4.8 Scenario 7 – Multiple Producers, Same Topic (S.07)

In this scenario, SA-1 and SA-2 both produce messages to a single topic owned by SA-3. This scenario is designed to test whether the proposed framework can correctly attribute cost away from the topic owner and toward the actual traffic generators. No discounts are applied.

Conduktor assigned an equal cost of 0.50 to each producer and zero to the topic owner (SA-3). In contrast, the proposed framework allocated the entire cost (1.00) to SA-3, without attributing cost to the producers. This shows a clear limitation that is discussed in section 7.5.5.

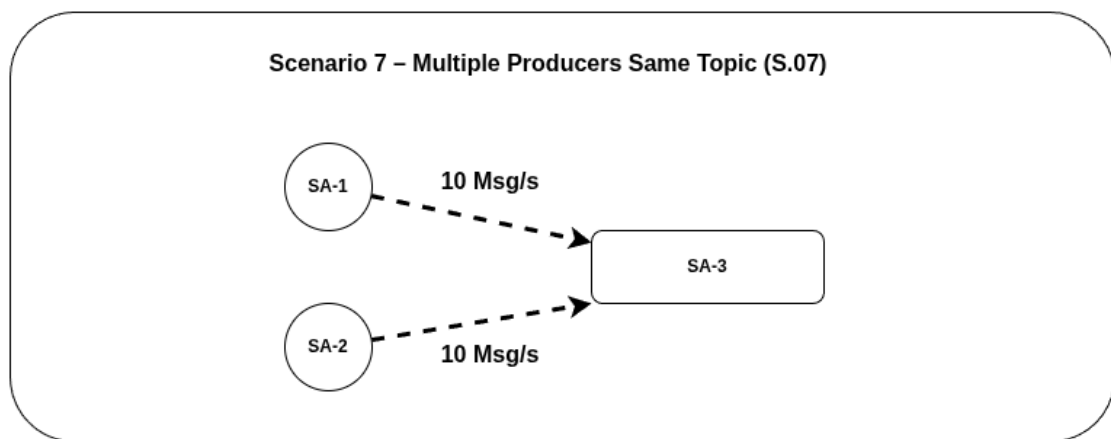


Figure 7.16: Scenario S.07 Diagram

Table 7.11: Scenario S.07 Allocation Results

System	Logical Unit	Base Allocation
Conduktor	SA-1	0.50
Conduktor	SA-2	0.50
Conduktor	SA-3	0.00
Proposed Framework	SA-3	1.00

7.5.4.9 Scenario 8 – Consumer Lag (S.08)

In this scenario, SA-1 sends messages to a topic also owned by SA-1 and consumed by SA-2 and SA-3. The SA-3 consumer group maintains near-zero lag, while the SA-2 consumer group exhibits high lag. This scenario tests if the system can reward low lag consumers. The goal is to validate whether the framework can encourage optimized consumption.

Conduktor equally divided the cost among all three service accounts, each receiving 0.33.

The proposed framework applied two discounts: D1 (High Topic Reader) and D3 (Low Consumer Lag). The D1 discount shifted the topic owner's allocated cost (SA-1) to the consumer groups (SA-2 and SA-3) in proportion to the number of messages each consumed. As SA-2 is slower due to the consumer lag, it commits fewer messages, resulting in less redistributed cost than SA-3.

Following that, D3 was applied to reward SA-3's low consumer lag. However, the current configuration of D3 redistributes the discount globally (using global redistribution), meaning that SA-1, the topic owner, was also burdened with part of the cost reduction intended for consumers. This leads to an unintended side effect: the producer is penalized despite having no influence over consumer lag performance. There's a clear need to refactor this discount and make the redistribution process impact only consumer groups.

The final allocations were: SA-1 with 0.92185, SA-2 with 0.04955, and SA-3 with 0.00665.

Although we observe that SA-3's efficient consumption behavior resulted in a reduced final allocation compared to SA-2, the overlap between D1 and D3 creates a contradictory dynamic. Low-lag consumers often consume more messages, which simultaneously triggers a higher redistribution from D1 and a reward from D3. This reduces the clarity and effectiveness of the discount applied.

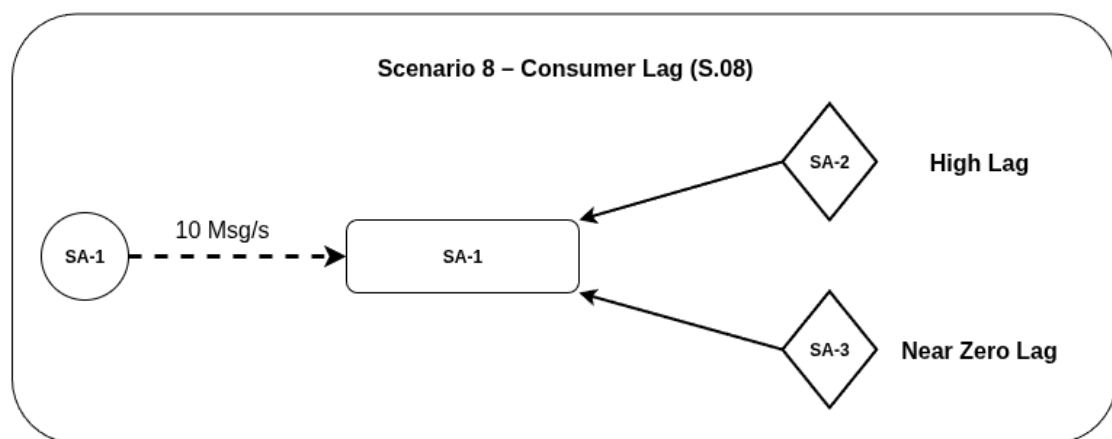


Figure 7.17: Scenario S.08 Diagram

Table 7.12: Scenario S.08 Allocation Results

System	Logical Unit	Base Allocation	After Discounts
Conduktor	SA-1	N/A	0.33
Conduktor	SA-2	N/A	0.33
Conduktor	SA-3	N/A	0.33
Proposed Framework	SA-1	0.9771	0.92185
Proposed Framework	SA-2	–	0.04955
Proposed Framework	SA-3	–	0.00665

7.5.4.10 Scenario 9 – Different Topics, One Consumer (S.09)

Three producers (SA-1, SA-2, and SA-3), each associated with a unique topic, publish independently. A single consumer, SA-4, consumes from all three topics. The goal of this scenario is to evaluate the attribution systems' ability to distribute part of the cost from topic owners to a cross-topic consumer. The consumer receives redistributed cost using D1, which shifts part of the load from topic owners to downstream consumers, reflecting shared data usage across topics.

Conduktor assigned 0.1667 to each producer and 0.4998 to SA-4. The proposed framework started with 0.33 per producer and, after applying the redistribution mechanisms, reduced their costs to 0.23 each, while assigning 0.31 to the shared consumer SA-4.

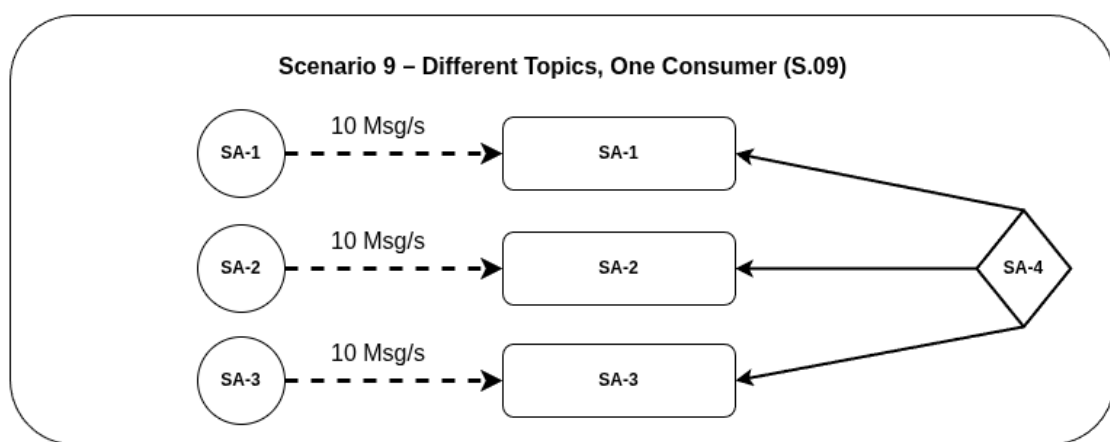


Figure 7.18: Scenario S.09 Diagram

Table 7.13: Scenario S.09 Allocation Results

System	Logical Unit	Before Discounts	After Discounts
Conduktor	SA-1	N/A	0.1667
Conduktor	SA-2	N/A	0.1667
Conduktor	SA-3	N/A	0.1667
Conduktor	SA-4	N/A	0.4998
Proposed Framework	SA-1	0.33	0.23
Proposed Framework	SA-2	0.33	0.23
Proposed Framework	SA-3	0.33	0.23
Proposed Framework	SA-4	–	0.31

7.5.5 Discussion

The comparative analysis reveals meaningful behavioral differences between the proposed cost allocation framework and the Conduktor Chargeback Interceptor across a variety of representative Kafka usage scenarios. These differences become especially pronounced in situations that

go beyond raw throughput measurement, highlighting the advantages of a more expressive and fairness-aware cost attribution model.

In the simplest scenarios, S.01 (Single Producer) and S.02 (Multiple Producers, Same Rate), both frameworks behave identically, allocating cost in direct proportion to resource usage. These cases serve as a baseline validation, confirming that the base allocation of the proposed system does not introduce unnecessary complexity in symmetric or isolated configurations. In such scenarios, the absence of consumers or differentiated storage usage means there is no need for rebalancing or corrective mechanisms, and both systems yield correct and identical outcomes.

However, from scenario S.03, where producers have differing levels of historical topic storage, divergence becomes apparent. While both SA-1 and SA-2 producers generate the same real-time message rates, only SA-1's topic includes a significant volume of preloaded data. The proposed framework correctly captures this historical load through the storage dimension, assigning a higher cost to SA-1. Conduktor, on the other hand, fails to detect this discrepancy and allocates costs evenly, overlooking an important component of infrastructure usage. This scenario demonstrates the proposed framework's sensitivity to different dimensions that impact cost.

Scenario S.04 introduces producers with different message rates. Both frameworks react to this asymmetry, but their respective behaviors differ in nuance. Conduktor allocates cost in a near-perfect 2:1 ratio between SA-1 and SA-2, reflecting a throughput-centric model. The proposed framework, by contrast, produces a more moderated distribution (approximately 1.5:1), due to the incorporation of multiple weighted dimensions: storage, compute, and network.

In scenario S.05, SA-1 acts solely as a producer, while SA-2 and SA-3 are consumers of the data published to SA-1's topic. The Conduktor system, consistent with its network attribution model, treats all three agents equally, assigning an allocation of 0.33 to each. This symmetric result stems from Conduktor's focus on ingress and egress metrics. Since all participants are transmitting or receiving similar data volumes, the system assumes equal contribution to the infrastructure load. In contrast, in the proposed framework, before the application of discounts, SA-1 bears the full cost (1.0), consistent with its ownership of the topic. However, through the application of two discount mechanisms, D1 (High Topic Reader) and D2 (High Out/In Ratio), the framework acknowledges the downstream utility of SA-1's data. These mechanisms shift a portion of SA-1's cost burden onto SA-2 and SA-3, resulting in a final allocation of 0.58 for SA-1 and 0.21 for each consumer (SA-2 and SA-3). This redistribution supports the Open Data principle outlined earlier: that teams who contribute widely used data should be financially rewarded, and consumers should share responsibility for the infrastructure costs they help generate. Nevertheless, it is important to acknowledge a potential counterargument: from a strictly infrastructural perspective, Conduktor's allocation may appear more fair. Since all three services are responsible for similar volumes of network traffic, one could argue that equal cost shares reflect proportional use. This criticism highlights a fundamental tension between infrastructure cost modeling and higher-level fairness policies. Our interpretation is that while the proposed framework takes a strong step in the right direction, by using higher-level fairness policies, the effect of the discount mechanisms in this scenario may be too low. Given that both consumers derive clear value from the data, a more

aggressive redistribution, perhaps by adjusting the discount percentages of each discount upward, would better reflect the collaborative dynamics at play. Increasing the discount effect would clearly reinforce the Open Data principle more expressively.

Scenario S.06 introduces schema governance as a differentiating factor. SA-1 and SA-2 behave identically in terms of message production, but only SA-2 enforces schema compatibility. While Conduktor fails to account for this behavioral nuance and produces symmetric allocations, the proposed framework applies a governance-based discount (D4), rewarding SA-2 by lowering its cost share. This illustrates the framework's capacity to reward good governance practices and incentivize adherence to standards that promote long-term data quality.

Conversely, scenario S.07 reveals a current limitation in the proposed approach. In this setup, SA-1 and SA-2 both produce to a topic owned by SA-3. Conduktor correctly allocates the cost to the actual message generators (SA-1 and SA-2), while the proposed framework attributes the entire cost to the topic owner (SA-3). This limitation arises from the fact that the proposed model relies solely on broker-side metrics, without direct visibility into producers and consumers. In contrast, Conduktor operates as a proxy, intercepting all Kafka traffic between clients and the broker. This architectural design allows it to associate network usage more precisely with the producing or consuming applications. However, this proxy model is not without its challenges. It introduces an additional network hop in the message flow, which may lead to increased latency, added operational complexity, and potential performance bottlenecks in high-throughput environments. In contrast, the proposed framework is fully decoupled from the Kafka brokers. It passively consumes metrics from existing monitoring systems such as Prometheus, avoiding any interference with Kafka's runtime operations. This architecture provides significant advantages in terms of scalability, maintainability, and ease of deployment. While this decoupling limits visibility into client-specific behavior, it ensures that the framework remains lightweight, non-intrusive, and compatible with a wide range of deployments.

Scenario S.08 further highlights a nuanced interaction between multiple discount mechanisms and their fairness implications. In this case, SA-1 is a producer, while SA-2 and SA-3 act as consumers of its topic. SA-3 consumes more messages and does so with near-zero lag, while SA-2 consumes fewer messages with higher lag. Conduktor, which lacks awareness of lag or consumer efficiency, allocates costs evenly among the three participants. The proposed framework applies two discounts: D1, which shifts part of the producer's cost to consumers in proportion to their usage, and D3, which rewards consumers that maintain low lag. The result is that SA-3 ends up penalized by D1 (due to consuming more messages) and rewarded by D3 (for maintaining low lag). However, these effects overlap in a counterproductive way: the more SA-3 consumes efficiently, the more it is both rewarded and penalized, leading to a cancellation effect that dulls the intended incentives. Furthermore, the D3 discount is currently redistributed using a global strategy (`_all`), which results in SA-1, the topic owner, absorbing part of the redistributed cost. This is misaligned with the principle of Accountable Consumption, as SA-1's behavior has no impact on consumer lag and should not be affected by it. These observations suggest that discount D3 should be scoped to redistribute only among consumers, not globally.

Finally, in scenario S.09, three producers (SA-1, SA-2, and SA-3) each publish to their own topic, while a single consumer, SA-4, consumes from all three. The goal is to test whether the systems can correctly shift part of the cost from producers to a cross-topic consumer who benefits from shared data. Conduktor assigns a higher cost to SA-4 (0.4998) and splits the remaining cost evenly across the producers (0.1667 each). This reflects its throughput-based logic, where the consumer's high network usage leads to a higher cost share. The proposed framework begins by assigning equal base costs to the producers (0.33 each) and none to the consumer. After applying the High Topic Reader discount (D1), the cost for each producer is reduced to 0.23, and SA-4 receives a cost of 0.31. This redistribution correctly reflects the role of SA-4 as a data consumer and supports the fairness goal of rewarding data providers. However, the shift is relatively small. SA-4 still pays less than the total cost borne by the producers, despite consuming from all three. This suggests that the discount effect is too weak. As in scenario S.05, increasing the discount percentage would make the redistribution more expressive and better align the allocation.

7.6 Threats to Validity

A central threat concerns the use of a simplified metric to represent compute usage. The use of the number of log segments metric per topic is intended to reflect broker compute consumption, but it captures only a narrow aspect of compute load and may overlook other significant factors. This is due to a clear limitation in correlating low-level compute metrics with logical units usage.

The simulator's abstraction of each service account as a single topic also introduces validity concerns. In practice, service accounts interact with many topics, each with distinct configurations and usage patterns. The one topic per account model used in the real-world scenario simplifies configuration and resource modeling but omits important nuances.

Another threat is the uniform use of fixed message sizes. A standardized 0.5 MB message size was chosen to reflect the production average, but real Kafka workloads often involve wide variability in message size and structure. This affects how storage, compute, and network resources are consumed. Additionally, all simulations assume a constant message production rate per producer, which does not reflect the bursty and often unpredictable traffic patterns observed in real environments. In production environments, message rates typically fluctuate based on application logic, batch intervals, user behavior, and external integrations, making the simulation's constant rate assumption a potential oversimplification.

Additionally, there is a discrepancy between the deployment topology used to derive the dimensional weights and the actual experimental setup. The weight calculations assume a production-scale cluster with three Kafka brokers, which reflects the structure used at Kuehne+Nagel and aligns with typical multi-broker deployments. However, the experiments themselves were conducted in a simplified single-broker environment to facilitate simulation control. This mismatch may distort the proportion of resource usage across dimensions, for example, compute and network behavior in a one-broker setup can differ significantly from that in a distributed, replicated

cluster. As a result, the experiments may not fully reflect the cost dynamics of the target deployment architecture.

Finally, the dimensional weighting in the cost model is subject to two distinct threats to validity. First, it relies on AWS-specific pricing assumptions, particularly with regard to EBS storage rates, EC2 instance costs, and inter-AZ network transfer fees. These assumptions reflect a specific cloud environment and may not generalize to on-premise deployments or alternative cloud providers with different cost structures. Second, the relative weights of the storage, compute, and network dimensions are inferred using the cost estimation formulas published in the Confluent blog post “A Guide to Mastering Kafka’s Infrastructure Costs.”. While these formulas provide a reasonable baseline, they are high-level approximations and may not accurately capture all operational nuances or cost drivers in every Kafka deployment.

In conclusion, these threats should be taken into account when interpreting outcomes or applying the approach in other settings.

Chapter 8

Conclusion

This final chapter consolidates the key insights from the dissertation, providing a summary of the framework’s contributions. Section 8.1 reflects on the main findings and discusses how the framework satisfies its design goals. Section 8.2 revisits the research questions introduced before. Finally, Section 8.3 outlines promising directions for future development.

8.1 Conclusions

The evaluation and comparative analysis presented in this thesis demonstrate that the proposed framework is capable of allocating costs in a fair way. The framework expresses cost differentiation across multiple infrastructure dimensions and introduces redistribution in a structured and interpretable way.

A key strength of the framework lies in its multi-dimensional weighted base allocation, which allows for decomposing the cost impact into different dimensions. Each dimension has a weight, reflecting the impact of that dimension on cost, and is linked to specific system metrics. For example, in Scenario S.03 (equal-rate producers with topics with different retained data), the framework correctly assigned a higher cost to the topic with larger historical storage. These results confirm that the framework can clearly account for how different resource types contribute to total cost. This is further supported by the real-world scenario, where the base allocation showed a strong positive correlation with infrastructure metrics: message rate ($\rho = 0.9577$) and log size ($\rho = 0.9169$). These findings validate that the base model reflects actual consumption behavior and successfully enforces the Accountable Consumption principle.

The discount mechanism introduces an additional layer of fairness logic. By enabling conditional reallocation based on observed behavior, such as downstream data reuse (Scenario S.05), schema enforcement (Scenario S.06), or consumer efficiency (Scenario S.08), the system is able to reward teams that contribute positively to the shared environment. These mechanisms allow the framework to reflect not just usage but value creation, aligning cost models with higher-order organizational principles.

A major design advantage of the framework is its adaptability across technologies and domains. While the evaluation focuses on Kafka, the underlying logic is technology-agnostic. The only essential requirements are (1) a configuration that defines dimensions and discounts tailored to the target system, and (2) access to meaningful metrics whose labels can be correlated with the logical units using regular expressions. This makes the framework applicable to a broad range of shared systems.

Crucially, this work emphasizes that observability is a prerequisite. The accuracy of the allocation process is fundamentally tied to the granularity and fidelity of the metrics available. The more precisely a system can expose who is consuming which resource, and under what conditions, the more accurate and fair the resulting allocation can be. In this sense, observability is the foundation upon which cost attribution stands. For example, in Scenario S.07, where two producers write to a shared topic owned by a third party, the framework could not distinguish between message generators and topic owners because the metrics lacked producer identity. This is not a flaw of the framework itself, but a consequence of insufficient observability.

8.2 Research Questions Revised

This chapter revisits the original research questions and provides answers based on the results and conclusions of the framework's evaluation.

RQ1: How can telemetry data be used to allocate infrastructure costs in shared environments?

Telemetry data provides a view of infrastructure resource usage. In the proposed framework, cost allocation is performed using resource usage by associating metric labels with logical units. These metrics are extracted from the monitoring layer, processed, and aggregated into cost allocations based on a configuration.

RQ2: How can resource usage be accurately mapped to organizational entities (e.g., teams, departments, or projects) using available metadata?

Resource usage can be mapped to organizational entities through the use of labels embedded in telemetry data, combined with regular expression-based mapping rules. This mapping is defined declaratively and enables flexible attribution even when naming conventions vary. However, the accuracy of this mapping depends on the consistency and traceability of these labels across the monitored system.

RQ3: What is the impact of considering multiple metrics with differentiated cost impact on the fairness of cost allocation?

Incorporating multiple metrics with differentiated weights allows the cost model to better reflect the actual cost drivers of the system. The evaluation in Scenario S.03 showed that the addition of

storage metrics enabled cost differentiation between two otherwise identical producers, enabling a fairer allocation.

RQ4: What approaches can be used to determine the relative contribution or importance of each metric in the allocation model?

The relative contribution of each dimension is highly dependent on the deployment context and the specific cost structure of the evaluated system. Different environments incur infrastructure costs in different proportions. Therefore, the weighting of each dimension in the cost model must be informed by a detailed analysis of the factors that drive actual cost in the target scenario. In this thesis, such an analysis was performed in Section 7.4.3.1.

RQ5: How can the cost allocation be extended to incorporate inter-team service relationships and incentivize efficient, collaborative infrastructure usage?

In this dissertation, incentivizing collaborative and efficient infrastructure usage was achieved through the design and application of a discount mechanism that shifts a portion of the cost allocation from one team to another, based on specific rules.

RQ6: What validation techniques and evaluation metrics can be applied to assess the fairness of the proposed cost allocation framework?

The framework was validated through a combination of synthetic scenarios and a simulation based on real production data. In the synthetic scenarios, it was benchmarked against an established industry solution (Conductor Chargeback Interceptor) across a diverse range of Kafka topologies and usage patterns. These controlled scenarios were designed to isolate specific behaviors, enabling a focused evaluation of the framework's ability to produce logically consistent and behaviorally fair outcomes. Fairness was assessed in accordance with the definition outlined.

8.3 Future Work

While the proposed framework has demonstrated flexibility and fairness in cost allocation, several areas remain open for improvement and extension.

First, the configuration of discount percentages is currently static and manually tuned. In Scenarios S.05 and S.09, the effect of the discount was often conservative, leading to weaker-than-intended redistribution. Future work should explore adaptive or metric-driven discount scaling, where the magnitude of each discount adjusts dynamically based on scenario-specific conditions.

Second, the framework could be extended to support label-specific regular expressions. Currently, a single regular expression is applied globally to extract ownership from all metric labels. In practice, different label types may follow different naming conventions. Allowing per-label regular expressions would improve mapping accuracy, especially in heterogeneous environments or organizations with inconsistent service naming.

Third, although the framework is designed to be technology-agnostic, its application has only been validated within Kafka. To confirm its generality, future work should apply the framework to other systems.

A further extension would be the introduction of the notion of hierarchy in the cost allocation. In practice, infrastructure usage often spans multiple organizational layers: A single team may contribute to multiple projects. Projects themselves may integrate multiple technologies. Supporting hierarchical relationships between entities would allow costs to be aggregated and analyzed at multiple levels, enabling better alignment with budgeting and accountability.

References

- [1] *ACM Digital Library*. <https://dl.acm.org/>.
- [2] *Amazon EBS Volume Types* — *aws.amazon.com*. <https://aws.amazon.com/ebs/volume-types/>.
- [3] *Apache Avro* — *avro.apache.org*. <https://avro.apache.org/>.
- [4] *Apache Kafka* — *kafka.apache.org*. <https://kafka.apache.org/>.
- [5] *Apache ZooKeeper* — *zookeeper.apache.org*. <https://zookeeper.apache.org/>.
- [6] *API Reference | Conduktor docs* — *docs.conduktor.io*. <https://docs.conduktor.io/platform/reference/api-reference/>.
- [7] Thijs Baars et al. “Chargeback for cloud services”. In: *Future Generation Computer Systems* 41 (2014), pp. 91–103. ISSN: 0167-739X. DOI: <https://doi.org/10.1016/j.future.2014.08.002>. URL: <https://www.sciencedirect.com/science/article/pii/S0167739X14001502>.
- [8] *Chargeback | Conduktor docs* — *docs.conduktor.io*. <https://docs.conduktor.io/gateway/interceptors/observability/chargeback/>.
- [9] *Cloud Compute Instances – Amazon EC2 Instance Types – AWS* — *aws.amazon.com*. <https://aws.amazon.com/ec2/instance-types/>.
- [10] *CloudZero*. <https://www.cloudzero.com/>.
- [11] *Conduktor - The Data Hub for Operational Data and AI* — *conduktor.io*. <https://conduktor.io/>.
- [12] *Conduktor overview | Conduktor docs* — *docs.conduktor.io*. <https://docs.conduktor.io/platform/>.
- [13] *Confluent | The Data Streaming Platform* — *confluent.io*. <https://www.confluent.io/>.
- [14] *Deep dive into Apache Kafka storage internals: segments, rolling and retention* — *strimzi.io*. <https://strimzi.io/blog/2021/12/17/kafka-segment-retention/>.
- [15] *Docker Compose* — *docs.docker.com*. <https://docs.docker.com/compose/>.
- [16] *DoiT*. <https://www.doit.com/>.
- [17] *FinOps Foundation - What is FinOps?* — *finops.org*. <https://www.finops.org/introduction/what-is-finops/>.
- [18] *FinOps Phases* — *finops.org*. <https://www.finops.org/framework/phases/>.
- [19] *Finout - FinOps Cloud Cost Management & Optimization Tool*. <https://www.finout.io/>.

- [20] FinOps Foundation. *FinOps Principles*. <https://www.finops.org/framework/principles/>.
- [21] FinOps Foundation. *Managing Shared Cloud Costs*. <https://www.finops.org/wg/identifying-shared-costs/>.
- [22] GitHub - prometheus/jmx_exporter: A process for collecting metrics using JMX MBeans for Prometheus consumption — github.com. https://github.com/prometheus/jmx_exporter.
- [23] GitHub - redpanda-data/kminion: Kminion is a feature-rich Prometheus exporter for Apache Kafka written in Go. — github.com. <https://github.com/redpanda-data/kminion>.
- [24] Global Infrastructure Regions & AZs — aws.amazon.com. https://aws.amazon.com/about-aws/global-infrastructure/regions_az/.
- [25] Grafana: The open and composable observability platform | Grafana Labs — grafana.com. <https://grafana.com/>.
- [26] Harness: AI-Native Software Delivery Platform — harness.io. <https://www.harness.io/>.
- [27] High-Performance Block Storage– Amazon EBS Pricing – Amazon Web Services — aws.amazon.com. <https://aws.amazon.com/ebs/pricing/>.
- [28] IEEE. <https://ieeexplore.ieee.org/>.
- [29] Kuehne+Nagel Homepage - Portugal — pt.kuehne-nagel.com. <https://pt.kuehne-nagel.com/pt/>.
- [30] Managed Kubernetes - Amazon Elastic Kubernetes Service (EKS) - AWS — aws.amazon.com. <https://aws.amazon.com/eks/>.
- [31] Overview | Conduktor docs — docs.conduktor.io. <https://docs.conduktor.io/gateway/>.]
- [32] Overview of Data Transfer Costs for Common Architectures | Amazon Web Services — aws.amazon.com. <https://aws.amazon.com/blogs/architecture/overview-of-data-transfer-costs-for-common-architectures/>.
- [33] Pricing — aws.amazon.com. <https://aws.amazon.com/ec2/pricing/>.
- [34] Prometheus - Monitoring system & time series database — prometheus.io. <https://prometheus.io/>.
- [35] Redpanda | High-performance streaming data platform — redpanda.com. <https://www.redpanda.com/>.
- [36] Schema Registry for Confluent Platform | Confluent Documentation — docs.confluent.io. <https://docs.confluent.io/platform/current/schema-registry/index.html>.
- [37] Science Direct. <https://www.sciencedirect.com/>.
- [38] Scopus. <https://www.scopus.com/>.
- [39] Simple Authentication and Security Layer - Wikipedia — en.wikipedia.org. https://en.wikipedia.org/wiki/Simple_Authentication_and_Security_Layer.
- [40] C. Spearman. “The Proof and Measurement of Association between Two Things”. In: *The American Journal of Psychology* 15.1 (1904), pp. 72–101. ISSN: 00029556. URL: <http://www.jstor.org/stable/1412159> (visited on 06/24/2025).

- [41] *Uncovering Kafka's Hidden Infrastructure Costs* — *confluent.io*. <https://www.confluent.io/blog/understanding-and-optimizing-your-kafka-costs-part-1-infrastructure/>.
- [42] *Vantage: Multi Cloud Cost Management & Optimization Tool* — *vantage.sh*. <https://www.vantage.sh/>.
- [43] *Welcome to The Apache Software Foundation* — *apache.org*. <https://www.apache.org/>.
- [44] *What is an Internal Developer Platform (IDP)?* — *internaldeveloperplatform.org*. <https://internaldeveloperplatform.org/what-is-an-internal-developer-platform>.

Appendix A

Cost Allocation Framework Configuration (JSON)

This appendix provides the JSON configuration file for the cost allocation framework. It defines logical units, metric dimensions, scrapers, weights, and discount mechanisms applied during cost attribution.

```
1 {
2   "port":8083,
3   "logic_unit_label": "team",
4   "logic_unit_relation":{
5     "extract_regex": "(?<=kn\\.)([^\.]*)",
6     "group_num": 1
7   },
8   "scrapers":[
9     {
10      "id": "prometheus",
11      "base_url":"http://prometheus:9090"
12    },
13    {
14      "id": "schemaRegister",
15      "base_url": "http://schema-registry:8081"
16    }
17  ],
18  "dimensions": {
19    "storage":{
20      "weight": 0.88,
21      "metrics":[
22        {
23          "weight":1.0,
24          "name": "kafka_log_size",
25          "aggregate_by_label": "topic",
26          "aggregate_by_operator" : "sum",
27          "default":0.0,
```

```

28         "scrapper_id": "prometheus"
29     }
30 ]
31 },
32 "compute":{
33     "weight": 0.12,
34     "metrics":[
35         {
36             "weight":1.0,
37             "name": "kafka_log_numlogsegments",
38             "aggregate_by_label": "topic",
39             "aggregate_by_operator" : "sum",
40             "default":0.0,
41             "scrapper_id": "prometheus"
42         }
43     ]
44 }
45 },
46 "discounts": [
47     {
48         "name": "high_topic_reader",
49         "trigger": "kminion_kafka_consumer_group_topic_offset_sum",
50         "discounted_label": "topic_name",
51         "penalized_label": "group_id",
52         "reallocation_strategy": "proportional",
53         "discount_percentage" : 0.1,
54         "scrapper_id": "prometheus"
55     },
56     {
57         "name": "valuable_topic_high_ratio_out_in",
58         "trigger": "kafka_server_total_bytes_out_per_sec_per_topic/
59             kafka_server_total_bytes_in_per_sec_per_topic",
60         "discounted_label" : "topic",
61         "discounted_condition": {
62             "compare_value" : 2.0,
63             "operator" : ">"
64         },
65         "penalized_label": "_all",
66         "reallocation_strategy": "equal",
67         "discount_percentage" : 0.3,
68         "scrapper_id": "prometheus"
69     },
70     {
71         "name": "valuable_consumer_group_low_consumer_lag",
72         "trigger": "kminion_kafka_consumer_group_topic_lag",
73         "discounted_label": "group_id",
74         "discounted_condition": {
75             "compare_value": 5,
76             "operator" : "<"

```

```
76         },
77         "penalized_label": "_all",
78         "reallocation_strategy": "equal",
79         "discount_percentage" : 0.1,
80         "scraper_id": "prometheus"
81     },
82     {
83         "name": "schema_governance",
84         "trigger": "compatibility_type",
85         "discounted_label": "topic",
86         "discounted_condition": {
87             "compare_value": "NONE",
88             "operator" : "!="
89         },
90         "penalized_label": "_all",
91         "reallocation_strategy": "equal",
92         "discount_percentage" : 0.1,
93         "scraper_id": "schemaRegister"
94     }
95 ]
96 }
```

Listing A.1: Cost Allocation Configuration (JSON)

Appendix B

Simulator Configuration Example (JSON)

This appendix includes an example JSON configuration file for the simulator. It specifies teams, producers, consumers, and Kafka topics used to model system workload in scenario S.05 in the comparative analysis.

```
1 {
2   "bootstrap_server": "localhost:9092",
3   "mode": "normal",
4   "teams": [
5     {
6       "id": "SA-1",
7       "topics": [
8         {
9           "id": "kn.SA-1.topic",
10          "num_partitions": 1,
11          "replication_factor": 1,
12          "preload_size": 0.0
13        }
14      ],
15      "consumer_groups": [],
16      "producers": [
17        {
18          "id": "kn.SA-1.producer",
19          "message_size": 0.5,
20          "message_rate": 10.0,
21          "distribution": "constant",
22          "topic": "kn.SA-1.topic"
23        }
24      ]
25    },
26    {
27      "id": "SA-2",
```

```
28     "topics": [  
29     ],  
30     "consumer_groups": [  
31         {  
32             "id": "kn.SA-2.consumer",  
33             "topics": [  
34                 "kn.SA-1.topic"  
35             ],  
36             "num_consumers": 1  
37         }  
38     ],  
39     "producers": [  
40     ]  
41 },  
42 {  
43     "id": "SA-3",  
44     "topics": [  
45     ],  
46     "consumer_groups": [  
47         {  
48             "id": "kn.SA-3.consumer",  
49             "topics": [  
50                 "kn.SA-1.topic"  
51             ],  
52             "num_consumers": 1  
53         }  
54     ],  
55     "producers": [  
56     ]  
57 }  
58 ]  
59 }
```

Listing B.1: Simulator Configuration Example(JSON)

Appendix C

Service Account Dependency Graph

This appendix presents the service account dependency graph used in the real-world scenario. It visualizes interactions between producers and consumers, helping illustrate logical dependencies within the shared infrastructure.

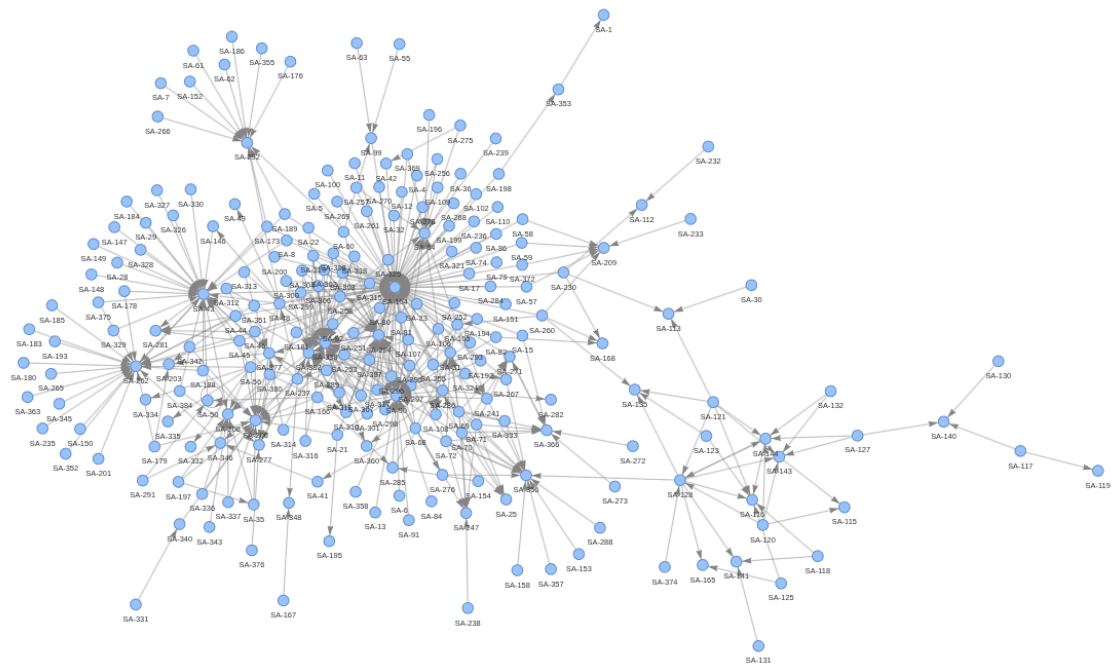


Figure C.1: Service Account Dependency Graph