Pontificia Universidad Javeriana Cali
Faculty of Engineering
Master's Degree in Software Engineering
Thesis

End-to-end model to apply FinOps for Evaluating Cost Efficiency of gRPC-based MACH Architectures: Resiliency, Reliability, and Security Trade-offs

William Martín Chávez González

Director: MSc Víctor Rodríguez
Co-director: Ph.D. Luisa Rincón

23rd Jul, 2026

Ficha Resumen
Trabajo de Grado Maestría en Ingeniería de Software

TíTULO: End-to-end model to apply FinOps for Evaluating Cost Efficiency of gRPC-based MACH Architectures: Resiliency, Reliability, and Security Trade-offs

1. Énfasis: Ingeniería de Software
2. Área de trabajo: FinOps - Cloud Architecture - Software Architecture
3. Tipo de proyecto (Aplicado, Innovación, Investigación): Aplicado
4. Estudiante: William Martín Chávez González
5. Correo electrónico: wchavez21@javerianacali.edu.co
6. Dirección y teléfono: Calle 18A No 56-20 - 3136549405
7. Director: Víctor Rodríguez, vhrz00@gmail.com
8. Vinculación del director: Planta
9. Correo electrónico del director: vhrz00@gmail.com
10. Co-Directora (Si aplica): Luisa Fernanda Rincón Pérez lfrincon@javerianacali.edu.co
11. Grupo o empresa que lo avala (Si aplica): No Aplica
12. Otros grupos o empresas: No Aplica
13. Palabras clave(al menos 5): FinOps, Unit Economics, Cloud Computing, Amazon Web Services (AWS), Cost efficiency, Total Cost of Ownership, Total Cost of Change (TCC), Frugal Architect, MACH Architecture, Microservices, gRPC, Architectural Characteristics, Resiliency, Reliability, Security, Affordability.
14. ODS que aplica al proyecto (Agenda 2030): 9
15. Fecha de inicio: 17 de junio de 2024
16. Resumen: Transitioning business operations to the cloud has resulted in cost efficiency outcomes that are below what was initially promised by the pay-as-you-go cloud computing financial model, a phenomenon that is currently demanding an adoption of a new finance-based mindset by Software Architects in order to become Frugal Architects that should design cost aware architectures on cloud, by treating costs as a crucial non-functional requirement. Therefore, FinOps has been conceived as a new cultural practice and operational model for Finance, Business, and IT personas to be conscious about the cost efficiency and the profitability of any architectural and infrastructural decision they make to operate and evolve services on cloud. In particular, IT departments have needed to perform cloud migrations by re-architecting their software systems for example, by using the Microservices architectural style as one of the most used approaches to take advantage of the cloud benefits; and that style is being recently aligned as well with MACH architectures (Microservices, API-First, Cloud Native and Headless) which allow the users to address business challenges by developing solutions using best-in-class services and technologies in a cost-effective and efficient manner. Furthermore, gRPC has played a

crucial role in building the interaction among Microservices, taking advantage of the HTTP/2 protocol and its resiliency, reliability, and security tactics; all of those non-functional requirements are considered crucial for those kind of services. Hence, this thesis offers an end-to-end integration model between FinOps, MACH Architectures and the gRPC Interaction Style for software architects to calculate the cost efficiency of their design decisions over gRPC-based Microservices operating on the Amazon Web Services cloud, so that they have a supportive tool in their path to become Frugal Architects.

Acknowledgments

First and foremost, I manifest my deepest gratitude to God, who has been a constant presence throughout my life, guiding me with His word, fortifying me during my most challenging times, and enveloping me in His mercy. It is unequivocally thanks to God that this work has come to fruition, and it is primarily dedicated to Him.

I express my deep gratitude to my parents, Betty González and Jaime Chávez, for their unwavering support throughout this academic undertaking, and particularly for the enduring family values they have instilled in me. Their exceptional discipline, kindness, love and commitment reach truly remarkable and unattainable levels.

Special acknowledgment is to the Pontificia Universidad Javeriana Cali for providing me with the invaluable opportunity and unwavering support throughout my academic journey as a student, especially to her Master's Degree in Software Engineering director, Luisa Fernanda Rincón, Ph.D, without whom I couldn't have finished this thesis; her support and patience were key in my most difficult struggles while developing this thesis, and her guidance helped me clarify my ideas to better orient them towards the goal. I deeply appreciate their recognition and care for me as an individual, allowing me to flourish fully. My sincere gratitude extends beyond the academic learning itself to the profound spiritual enrichment gained through the understanding of their principles such as *"To serve is always to be reborn"*, *"In everything, love and serve,"* and above all, *"To be more to serve better"*. I am also deeply grateful with the San Francisco Javier Pastoral Center that provided me with the invaluable opportunity to contribute to God's work within communities through the mission camp, fostering my growth as individual, and guiding me towards the truth under God's direction, with particular appreciation for the Vise-Rector, Father Luis Alfonso Castellanos, the Director, Yuliana Roldan Giraldo, and Deicy Vargas Triviño.

Special and heartfelt thanks go to Alessia A., Mayra Alejandra Rengifo, Alexánder Valencia Altamirano, Andrea Chittano, Debora Zanaboni, and Martina De Matteis, with whom I felt emotionally supported and who also contributed through their insightful perspectives and points of view to validate the business CANVAS model about a music therapy ecosystem that I presented as the final project for the Software Business course, and for the Diploma in e-commerce course.

I appreciate also my thesis directors, Luisa Fernanda Rincón and Víctor Rodríguez for the enlightenment they offer me during the cloudiest periods of this thesis, when I needed guidance and direction. I extend my gratitude to my Telematics Engineering thesis director, Daniel Barragán Calderón, with whom I had previously work on the prequel of this thesis back in 2017 and who provided feedback from his DevOps expertise as well in this one.

Finally, I would like to specially thank to the FinOps Professionals, and to my co-workers in the company I worked with since I started my thesis, who also contributed with feedback to the end-to-end model and to the tool for Software Architects delivered as the results of this thesis, specially to the Principal Java Software Architect, Massimo Montefusco.

In memory of

Álvaro Pachón De La Cruz, Ph.D.
Ex Head of the Department of Computing and Systems, Icesi University.

A teacher, a leader, and an example who taught us **to dignify our profession**.

I remember your guidance with deep gratitude. Rest in peace; may God keep you in His glory.

Abstract

Transitioning business operations to the cloud has resulted in cost efficiency outcomes that are below what was initially promised by the pay-as-you-go cloud computing financial model, a phenomenon that is currently demanding the adoption of a new finance-based mindset by Software Architects in order to become frugal architects who design cost efficient architectures in the cloud, treating costs as a crucial non-functional requirement. Therefore, FinOps has been conceived as a new cultural practice and operational model for Finance, Business, and IT personas to be conscious of the cost efficiency and the profitability of any architectural and infrastructural decision they make to operate and evolve services on the cloud. In particular, IT departments have needed to perform cloud migrations by re-architecting their software systems using the microservices architectural style to take advantage of the cloud benefits ; this style is also being recently aligned with MACH architectures (Microservices, API-First, Cloud Native, and Headless), which allows users to address business challenges by developing solutions using best-in-class services and technologies in a cost-effective and efficient manner. Furthermore, gRPC has played a crucial role in building the interaction among microservices, taking advantage of the HTTP/2 protocol and its resiliency, reliability, and security tactics; all of these non-functional requirements are considered crucial for those kinds of services. Hence, this thesis offers an end-to-end integration model for software architects to calculate the cost efficiency of their architectural design decisions over gRPC-based microservices operating on the Amazon Web Services cloud, so that they have a supportive tool in their path of becoming frugal architects.

Keywords: FinOps, Unit Economics, Cloud Computing, Amazon Web Services (AWS), Cost efficiency, Total Cost of Ownership (TCO), Total Cost of Change (TCC), Stock Keeping Unit (SKU), Frugal Architect, MACH Architecture, Microservices, gRPC, Architectural Characteristics, Resiliency, Reliability, Security, Affordability.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

"Roles will evolve to meet the needs of FinOps. More polly-skilled employees who have a business head, fiscal thinking, and technology acumen will emerge. Finance people will learn cloud, just as IT people will learn finance.
Think back to when the idea of full-stack engineers was new; now it's time to start thinking about full-business people."

J.R. Stortment & Mike Fuller, Cloud FinOps, 2023
(Storment & Fuller, 2023)

Cloud vendors have accelerated businesses by providing scalable infrastructure solutions and by offering a ***pay as you go* financial model** (Amstrong, 2020). As highlighted by Storment and Fuller (2023), this consumption-based service model (similar to the utilities one) is well understood by finance teams, even though it implies a different sourcing process and granularity. Therefore, software architects operating cloud-native software systems are concerned about **making cost a non-functional requirement** (Vogels, 2023b). However, how could software architects make design decisions to ensure the cost-efficiency of their solutions in the cloud?

In order to answer this question, it is necessary to dive deep into how technology and operations have evolved to promote the benefits of cloud computing. On one hand, in terms of **Software Architecture**, since the cloud business was conceived, the role that **microservices architectural style** has been playing is crucial (Amazon Web Services, 2024e) because, compared to monolithic applications, it promotes agility, flexible scaling, easy deployment, technological freedom, reusable code, and **resiliency** (Amazon Web Services, 2024e). Those arguments have been recently reinforced by **MACH's pillars** (MACH Architecture, 2021a) the first of which *"comprises Microservices for independent deployment of functionalities"* (MACH Architecture, 2021b) in its purpose of setting *"principles, patterns, and associated technologies that enable adopters to solve business problems by composing solutions using best-in-class services and technologies"* (MACH Architecture, 2021b), with economic efficiency (cost-effective scaling and optimal resource utilization) being one of its benefits.

Moreover, microservices could implement different interaction styles for granting inter-service communication, one of which is called gRPC, which is a communication protocol and high-performance RPC framework introduced by Google (Babal, 2024). In fact, *"with the advent of Microservices, the adoption of gRPC is exponentially growing,"* as mentioned by both Indrasiri and Kuruppu (2020), and the InfoQ architectural trends report published in 2023 (Betts, 2023).

On the other hand, in terms of the responsibility of **Software Architects** —as well as Cloud Architects—who daily need to grant different non-functional requirements for their services (Bass, Clements, & Kazman, 2021), deploying those services in the Cloud implies having a strategy to deliver the business value that has been promised in operations. In order to do so, as suggested by Dr. Werner Vogels in the Keynote he presented in the AWS re:Invent 2023 event (Vogels, 2023a), the Software Architect needs to become a **Frugal Architect** by following a set of *"simple laws for building cost-aware, sustainable, and modern architectures"* the first of which is ***Make Cost a Non-functional Requirement*** (Vogels, 2023b). Moreover, in the *Architecting for Cloud* article (FinOps Foundation Project a Series of LF Projects, 2024a), **FinOps** (Finance + DevOps) has already defined a set of functional activities, with their respective inputs and outputs, that are expected to be distributively assigned to the FinOps personas - like **Finance People**, **Engineers**, **Executive and Leadership**, and **Procurement and Sourcing People** (Storment & Fuller, 2023) - for *"designing and modernizing solutions with cost-awareness and efficiency to maximize business value while achieving performance, scalability, and operational objectives"* (FinOps Foundation Project a Series of LF Projects, 2024a). In fact, as mentioned by Storment and Fuller (2023), *"roles will evolve to meet the needs of FinOps. More poly-skilled employees who have a business head, fiscal thinking, and technology will emerge. Finance people will learn cloud, just as IT people will learn finance"*. Moreover, *"think back to when the idea of full-stack engineers* was new; now it's time to start thinking about full-business people".

FinOps's main goal is to offer a new cultural practice to define how **FinOps personas** can reduce their communication gaps in order to align technological design decisions and solutions with financial and business goals (cost efficiency and profitability) in an inform, optimize, and operate cycle (Storment & Fuller, 2023). Moreover, regarding cloud operations, the FinOps Foundation (2025) has defined **cloud unit economics** to measure the cost-efficiency and profitability of cloud architectures. However, there has not been a descriptive and practical specification, up to now, of how to adapt this process, its inputs, outputs, and unit economics to evaluate the cost-efficiency of software architectures in general, and for gRPC-based microservices on cloud vendors in particular. Therefore, the purpose of this thesis is to present a comprehensive **end-to-end model** (from high-level requirements specification to a FinOps application) that calculates its proposed cost-efficiency concerns as a key **architectural characteristic** in the design of high-quality gRPC-based microservices on a cloud vendor by following MACH principles and patterns to ensure economic efficiency. Finally, this end-to-end model is expected to be used by Software Architects as a supporting tool in their path to becoming Frugal Architects (Vogels, 2023b).

1.1 Problem Definition

The adoption of the cloud by corporations worldwide has been growing intensively since 2019 (COVID-19 lockdown) (Moorhead, 2020). However, when enterprises run workloads and data sets using traditional infrastructure patterns, such as business applications that process and store data the same way they did when on-premises, there is a negative cost impact to using a public cloud, as confirmed by the Infoworld article that highlights the retirement of this cloud operating and financial *pay as you go* model (Linthicum, 2024). The Infoworld article also adds: *"In other words, those who attempted to use the cloud as a simple host for their workloads and took no steps to optimize those workloads for their new location had much larger bills than expected"* (Linthicum, 2024). As a result, corporations that operate in the cloud have recently begun to be concerned about the adoption of the **FinOps cultural practice** as a key success factor, according to the 2022 State of FinOps data (FinOps Foundation Project a Series of LF Projects, 2022) (Storment & Fuller, 2023).

FinOps promotes collaboration between business stakeholders, financial managers, procurement teams, and engineers so that they can improve the cost-efficiency of those solutions by designing **cost aware architectures** (FinOps Foundation Project a Series of LF Projects, 2024f), and elaborating a cost strategy to achieve the highest possible Return on Investment (ROI). In fact, the FinOps Foundation has already defined a set of functional activities, with their respective inputs and outputs, that are expected to be distributively assigned to the FinOps personas for *"designing and modernizing solutions with cost-awareness and efficiency to maximize business value while achieving performance, scalability, and operational objectives"* (FinOps Foundation Project a Series of LF Projects, 2024a). Moreover, regarding cloud operations, FinOps defines **unit economics** (FinOps Foundation, 2025) to measure the cost efficiency of cloud architectures.

On the other hand, since the cloud business was conceived, the role that **microservices architectural style** has played in the architectural design of its services is crucial, considered one of the building blocks of the MACH Architecture that *"proposes a set of principles, patterns, and associated technologies that enable adopters to solve business problems by composing solutions using best-in-class services and technologies"* (MACH Architecture, 2021a), one of which establishes the creation of apps and services *"to leverage the full capabilities of the cloud computing delivery model, designed, developed, and managed in the cloud and for the cloud"* (MACH Architecture, 2021a). Furthermore, MACH Architecture, compared to Monolithic one, *"enables the use of the best technology for each service, as they operate independently"* (MACH Architecture, 2021b) as a result of adopting microservices. Moreover, *"with the advent of microservices, the adoption of gRPC is exponentially growing"* as mentioned in both (Indrasiri & Kuruppu, 2020), and in the InfoQ architectural trends report published in 2023 (Betts, 2023). Each communication pattern implemented by gRPC solves different inter-process communication problems and use cases (Indrasiri & Kuruppu, 2020). In particular, to improve the quality of those interactions, trade-offs such as **resiliency, security, and reliability** should be analyzed because architectural decisions, i.e., switching from one communication pattern to another, also imply architectural decisions in the cloud infrastructure to support those architectural characteristics. Those cloud architectural

decisions result in costs that should be uniformly calculated and forecasted by software architects and engineers.

In the particular context of the AWS Cloud, the **Well Architected Framework's** Operational Excellence, Performance Efficiency, **Reliability, Security, Cost Optimization**, and Sustainability pillars (Eliot & Valverde, 2018) provide a solid foundation to start thinking about how to make design decisions to improve cost efficient solutions. However, they do not specify implementation guidelines for each business scenario or technological approach. Moreover, there has not been a descriptive and practical specification, up to now, on how to adapt FinOps, its inputs, outputs, and KPIs in order to evaluate the cost-efficiency of software architectures in general, and for RPC-based microservices of a MACH Architecture deployed on any cloud vendor (nor for AWS in particular). Unfortunately, there isn't any reference available specifying how different architectural design decisions (tactics, patterns, and strategies) affect the synergy between resiliency, reliability, security, and cost-efficiency of those microservices.

Finally, if software architects need to justify their design decisions about using one gRPC communication pattern or another in a microservices architecture with a cloud vendor in terms of costs, they do not have a formalized model for evaluating costs in the design phase to avoid taking empirical and non-formalized approaches that might lead to overspending issues and, therefore, a lack of trust from the **Financial teams**. Therefore, this thesis answers the following questions:

1. How can resiliency, security, and reliability tactics for gRPC-based microservices be mapped onto a cloud vendor's architecture and integrated with FinOps operational tactics to promote cost efficiency?
2. What is the process for creating an end-to-end cost model for Software Architectures (e.g., cloud-based gRPC microservices architecture), and what are the key inputs and outputs, taking into account deployment variants?
3. How can the proposed end-to-end cost model be developed into a convenient tool for software architects working in a cloud environment?
4. How can the perceived usefulness of the tool be aligned with Software Architects' expectations for calculating cost-efficiency in MACH architectures?

1.2 Project Goals

1.2.1 Main Goal

To propose an end-to-end model that applies FinOps to evaluate the cost efficiency of gRPC-based MACH architectures and the design decisions that impact the cloud drivers affecting costs.

1.2.2 Specific Objectives

1. To specify the Protocol Buffer files corresponding to the four business use cases while accurately reflecting the required gRPC communication patterns for use as primary inputs in the cost estimation model.
2. To map the resiliency, security, and reliability tactics recommended by software architects when

implementing software architectures in AWS, integrating them with FinOps operational tactics to improve operational cost-efficiency and incorporating them as part of the model's input parameters.

3. To design the end-to-end model for determining the cost of gRPC-based MACH Software Architectures, including its inputs, outputs, and unit economics.
4. To develop the proposed end-to-end model into a practical tool for software architects operating in the cloud.
5. To conduct a qualitative evaluation of the proposed cost estimation model, focusing on perceived usefulness, intention to use, and usability based on expert feedback from professionals in cloud architecture and software design.

1.3 Scope

In order to design and tune the model, it was necessary to specify a set of **Protocol Buffer** files related to the desired gRPC-based microservices. By following MACH principles and patterns along with FinOps, cost efficiency is achieved; this is the reason why other architectural styles or patterns will not be considered part of the scope of this project.

In particular, for the baseline, the Protocol Buffer files include the following most-common Business Use Cases (BUC) supported by gRPC in the context of E-Commerce systems (Indrasiri & Kuruppu, 2020), including transactionality, booking, retries, and rollback (Babal, 2024) for the four gRPC communication patterns. Moreover, even though GDPR compliance is key in the data protection shared by those microservices, this is not to be regarded as the scope of this project and should be considered future work for evaluating cost-efficiency in production environments. Therefore, this project only emphasizes the implications of gRPC communication patterns for E-commerce business use cases, along with the trade-offs of resiliency, reliability, and security (without considering policies). The following are the security patterns and tactics that are part of the scope: TLS Handshake (Babal, 2024), Certificate generation with gRPC TLS credentials (Babal, 2024), and OAuth 2.0 and JWT (Indrasiri & Kuruppu, 2020).

On the other hand, regarding objective 2, the mappings are documented in an Excel matrix attached to this project, considering the definitions of both Reliability and Security from **ISO/IEC 25010:2023** (ISO/IEC JTC 1/SC 7, 2023a); and Resiliency defined by Firesmith (2020) in the Software Engineering Institute (SEI) blogs for Software Architecture, in order to map the **trade-offs** between them and cost-efficiency (FinOps Foundation Project a Series of LF Projects, 2024b).

Moreover, the practical tool asks for high level requirements specifications about gRPC-based microservices in the same E-Commerce context. The outcome is the estimated Total Cost of Ownership per month and year, along with basic recommendations for ensuring cost efficiency from a FinOps perspective. The practical tool offers a list of options to customize a basic E-Commerce microservice that covers only the four Business Use Cases presented in the first objective, as well as options for including the tactics and patterns covered by the mappings done in Objective 2. Once a set of options is selected, FinOps strategies for ensuring cost efficiency are also presented for selection (Savings Plans and Reserved Instances). Finally, an estimation of cloud costs for the

AWS cloud vendor is provided based on the AWS Pricing Calculator (Amazon Web Services, 2025a), considering the operation of the architecture over 1 month and 1 year (FinOps strategy costs, such as Savings Plans and Reserved Instances, can be estimated only for 1 year or 3 years in AWS). The following were the restrictions for building this tool in terms of the number of parameters to be selected to enable customization: four business use cases defined in the Protocol Buffers for each gRPC communication pattern to be supported by the Architecture. 2 reliability tactics: **client side load balancing** and **server side load balancing**. 3 resiliency patterns: **Timeout**, **retry**, and **circuit breaker**. 2 security tactics: **TLS certificates** and **OAuth 2.0 with JWT tokens**.

On the other hand, in terms of FinOps, the project focuses solely on evaluating the cost-efficiency of each business use case. Profitability and ROI are included in the project's scope, conditioned by the specification of the monthly payment received by a customer in a month to consume a service; this way, the tool can simulate that to approximate ROI. Finally, considering FinOps as an operational model that is agnostic to cloud vendors, some of which are among the most relevant today (AWS, Microsoft Azure, and Google Cloud Platform (GCP)), the cost reports are the only task considered part of the scope of the proposed model. However, as future work, its variants should be evaluated for cloud vendors other than AWS, which has been selected based on the criteria of being part of the area of work experience of the executor of this project.

1.4 Justification

As a consequence of implementing **FinOps**, the establishment of a precise criterion for assessing cost-efficiency within any **architectural style**, and particularly for **gRPC communication patterns**, holds significance within the realm of **Software Architecture**, and in MACH Architectures in particular. Applying FinOps practices in MACH Architectures is crucial because, by following **MACH principles and patterns** along with FinOps, economic efficiency is achieved. This is considering that the Total Cost of Ownership (TCO) calculation is evolutionary in these architectures, taking into account the **Total Cost of Change (TCC)** (MACH Alliance, 2023a). Such delineated criteria are fundamental for **Software Architects and Engineers** to make conscientious architectural and technical decisions, taking into account their direct impacts on the business, the corresponding responsibilities assigned to architects, the imperative need to reconcile various quality attributes, as well as the resulting implications on costs: "When our infrastructure is hosted in the cloud, we need to be able to estimate the cost of new projects, to make this factor into our decision-making, as costs can make or break a project if not justified" (San Miguel, 2024).

On the other hand, working in gRPC-based MACH Architectures, apart from being one of the most innovative technologies for inter-service communication, is key to the industry evolution in granting high-performance microservices solutions.

Furthermore, architects understand the key rule: everything is a **trade-off**. For **gRPC-based microservices** , it is crucial to have **resiliency, reliability, and security tactics and patterns** implemented to improve the quality of this style. Therefore, Software Architects and their teams could count on a tool that helps them estimate those trade-offs and justify their technical decisions or selections within an execution budget defined by CFOs, Financial Managers, and

Project Managers. This approach enhances the competitiveness of Software Architects and enables enterprises to improve both quality and business value (Return on Investment (ROI)).

Moreover, one of the key concerns for CFOs is to determine an appropriate risk-return trade-off *"to maximize the market value of the firm for its shareholders. The risk-return decision will influence not only the operational side of the business (capital versus labour) but also the financial mix (stocks versus bonds versus retained earnings)"* (Block, Hirt, & Danielsen, 2017). They also have a **trade-off** between profitability (ROI) and risk, and the goal is to maximize shareholder's financial health (Block et al., 2017). On the other hand, *"it's no longer acceptable for teams to consider only their own priorities"* (referring to Financial and Technical operations teams). *"If the technical operations team doesn't take into account the impacts of their cloud spending, finance will have an impossible challenge of forecasting and budgeting the organization's cloud spend. Alternatively, if finance takes full control of cloud spending and requires approvals for every resource, the organization will struggle to take advantage of the speed and agility of having variable spend resources and on-demand infrastructure"* (Storment & Fuller, 2023). Hence, FinOps is expected to be integrated into digital transformation in the cloud; otherwise, the value of the cloud, in terms of *"speed of innovation"*, could be lost (Storment & Fuller, 2023).

Finally, for both IT and Financial teams, this end-to-end model will serve as a transparent point of reference for granting financial health and promoting the AWS WAF's cost optimization pillar (Amazon Web Services, 2023a, 2023b) in the execution and operation of software projects that implement **gRPC-based microservices on a MACH Architecture operated** (Babal, 2024; Indrasiri & Kuruppu, 2020) in the AWS Cloud. These technologies (gRPC, MACH, and AWS) were chosen to enable the presenter of this project proposal to align with the business context in which he has been working for the last five years.

1.5 Methodology

FinOps provides emergent practices to solve cost-efficiency issues for Cloud Architectures (FinOps Foundation Project a Series of LF Projects, 2024a), and these practices are not formally specified for software architectures, nor for gRPC-based microservices in particular (Babal, 2024). This project is located in the complex area of the Cynefin framework (Snowden, 2005), where validations via **Proof of Concepts (POCs)** were planned to be executed in a short time frame to fail fast and obtain results quickly enough to reaffirm the current direction or take another one if necessary, as well as to validate which expected results would not be delivered on time. Regarding this concern, this approach was used in this project to ensure the end-to-end model and the **incremental deliveries** of the practical tool to validate the model at each milestone: the business use cases and architectural tactics and patterns to be covered in the first milestone; the interaction with the end-to-end model's activities at the end of objective 3; the improvements to the practical tool based on the adjustments that might be needed after objective 3; and the user mechanisms to evaluate the perceived usefulness of the model. This might help validate in the early stages if it would be possible to have an End-to-End model at the conclusion of the investigation, based on the risks that might be detected during the project implementation, and how to adjust and adapt the activities

to mitigate those risks in time to reach the main goal at the end of the last milestone.

In order to accomplish the main result, the well-known collaboration protocol called **Objectives and Key Results (OKRs)** (Doerr, 2019) was implemented. On one hand, an **Objective** indicates only what should be achieved. On the other hand, the **Key Results** indicate how that Objective must be achieved, and they need to be specific, measurable, timely, aggressive, and realistic at the same time (Doerr, 2019). Hence, the following were the OKRs that were accomplished in this project for each Specific Objective.

1. **Specific Objective (01)**: To specify the Protocol Buffer files corresponding to the four business use cases, accurately reflecting the required gRPC communication patterns for use as primary inputs in the cost estimation model, in 1.5 weeks: **BUC1**: **gRPC Unary pattern**: Retrieve orders by using an order ID from the client to the server. **BUC2**: **gRPC Server Streaming pattern**: The business needs to retrieve all possible orders that match a search criterion (term or filter). **BUC3**: **gRPC Client Streaming pattern**: Update a set of orders. **BUC4**: **gRPC Bi-Directional Streaming pattern**: Send a continuous set of orders and process them into combined shipments based on the delivery date.
2. **Specific Objective (O2)**: To map the resiliency, security, and reliability tactics recommended by software architects when implementing software architectures in AWS, integrating them with FinOps operational tactics to improve operational cost-efficiency, and incorporating them as part of the model's input parameters:
   (a) **KR1**: Map the **four communication patterns** defined by gRPC: Unary or Simple, Client-Streaming, Server-Streaming, and Bi-Directional Streaming (Indrasiri & Kuruppu, 2020).
   (b) **KR2**: Map the four trade-offs as a result of comparing RESTful versus each of the four gRPC communication patterns.
   (c) **KR3**: Map **2 reliability tactics**: Server-side and Client-side load balancing (Babal, 2024).
   (d) **KR4**: Map **3 resiliency patterns**: Timeout pattern, Retry pattern, Circuit Breaker pattern; and Error Handling tactic (Babal, 2024).
   (e) **KR5**: Map **2 security tactics**: **TLS certificates** and **OAuth 2.0 with JWT tokens** (Indrasiri & Kuruppu, 2020).
   (f) **KR6**: Map FinOps operational tactics for the four variations resulting from implementing the reliability, resiliency, and security tactics.
   (g) **Specific Objective (O3)**: To design the end-to-end model for determining the cost of gRPC-based MACH Software Architectures, including its inputs, outputs, and unit economics:
      i. **KR1**: Define an operative baseline with the interfaces definition (Protocol Buffers) related to E-Commerce backend microservices, whose payload calculation should be determined to estimate the cost variation or Total Cost of Change (MACH Alliance, 2023a) (**marginal cost**), generated as a result of applying reliability, resiliency, and security patterns and tactics.
      ii. **KR2**: Define 3 key inputs and 3 key outputs, including cloud billing calculations and

FinOps adaptation.

iii. **KR3**: Define the activities that should be part of the process: activities related to ASR mapping (Bass et al., 2021), another one related to FinOps strategies (Storment & Fuller, 2023); and regarding cost reports.

(h) **Specific Objective (O4)**: To develop the proposed end-to-end model into a practical tool for software architects operating in the cloud. **KR**: one practical tool. In order to achieve that on time, I proposed an application of 5 increments:

i. **Increment 1:** Protocol Buffer message request and response size calculator and Networking costs calculator.

ii. **Increment 2:** Upfront and Total Lifecycle Forecasted Costs Calculator - Reliability and Resiliency tactics.

iii. **Increment 3:** Upfront and Total Lifecycle Forecasted Costs Calculator - Security and microservices tactics.

iv. **Increment 4:** Upfront and Total Lifecycle Forecasted Costs Calculator - Scalability and Cloud tactics.

v. **Increment 5:** Upfront and Total Lifecycle Forecasted Costs Calculator - Containerized Environment factors.

(i) **Specific Objective (O5)**: To conduct a qualitative evaluation of the proposed cost estimation model, focusing on perceived usefulness, intention to use, and usability, based on expert feedback from professionals in cloud architecture and software design.

i. **KR1**: To conduct and document a qualitative evaluation of the proposed cost estimation model.

ii. **KR2**: To conduct and document a qualitative evaluation of the proposed practical tool, focusing on perceived usefulness, intention to use, and usability, based on expert feedback from professionals in cloud architecture and software design.

iii. **KR3**: Document the results.

The criteria to be evaluated in the proposed end-to-end model are based on the cost variation (or marginal cost) after adding either a business use case, architectural tactic, or pattern to the desired architecture, and on the parameters included in the scope of the practical tool. Therefore, the evaluation estimated the cloud **Total Cost of Ownership (Upfront costs and lifecycle Forecasted Costs derived from the FinOps strategy)** for each variation selected in the tool as a estimation of implementing the desired architecture in AWS with an operational baseline of 1 month. A brief explanation of how those architectural decisions impact resource units and monetary value was provided as part of the FinOps operational tactics parameters' description in the same tool. Therefore, the methodology was adjusted to an incremental one that started from the baseline (the Protocol Buffers definition) and ended with both the qualitative evaluation of the proposed tool and the cost-estimation model, both of which were documented and presented to the Master's Degree judges.

Finally, as part of the teamwork with the Thesis's Directors, a one-hour weekly meeting was scheduled to follow up on the planning and direction as can be borne out by the Gantt chart that has been attached in the file titled **"Attachment-Planning-Gantt Chart".**

1.6 Results

As a result, we were able to evaluate the End-to-end model with the tool for Software Architects. For the FinOps Inform phase, the complexity of the Total Cost of Change (marginal cost) of each architectural tactic or pattern was calculated for each architectural characteristic: security, resiliency, and reliability; as well as for the Amazon Web Services that help implement those tactics and patterns for MACH Architectures.

Moreover, it was possible to establish a comparison between the base costs, defined as the networking costs in the AWS cloud associated with the protocol buffer files of each business use case, and the marginal cost associated with each additional AWS service, software tactic, or pattern in the tool by using the AWS Pricing calculator. Therefore, Software Architects can be aware of this marginal cost in comparison with the minimum requirement that needs to be satisfied by operating one microservice, which is the cost linked to its interaction style or gRPC communication pattern: unary, server streaming, client streaming, and bi-directional streaming.

Furthermore, some basic **FinOps** strategies were manageable within the model and the tool to optimize the Total Cost of Ownership calculation for a gRPC-based Microservice. It was possible to estimate **unit economics**, especially those related to the Customer Acquisition Cost (CAC) per month (marginal cost), understood for the scope of this project as costs per user, and the monthly Return on Investment (ROI) by introducing an Average Revenue per User (ARPU) per month as an input for the cost-efficiency calculator tool.

Finally, it was possible to estimate the same unit economics for a complete portfolio of gRPC-based microservices, which represents the MACH Architecture. This was made possible by extending the end-to-end model to be applied at an architectural landscape level, not only at the service level.

Chapter 2

Reference Framework

"The impediment to action advances action. What stands in the way becomes the way."

Marcus Aurelius.
Cited by Holiday (2014)

2.1 Theoretical Framework

2.1.1 ISO/IEC 25010:2023

The ISO/IEC 25010:2023 is a crucial reference for quality attributes that were defined as a consensus for ensuring the quality of software systems. It *"defines a product quality model, which is applicable to Information and Communication Technology (ICT) products and software products.* It is composed of nine characteristics, each of which is subdivided into sub-characteristics, and together they build a reference model to specify, measure, and evaluate quality attributes (ISO/IEC JTC 1/SC 7, 2023a).

As part of its characteristics, **Reliability** is an architectural characteristic that could include availability, data integrity, among others (Ciceri et al., 2022). In addition to that scope, Erder (2021) explains that to achieve reliability, the software system must be highly available and operate correctly. **Reliability** tactics in Microservices are well known: **Circuit Breaker** (Richardson, 2019) and **Retry** (Bass et al., 2021).

Unfortunately, the **ISO/IEC 25010:2023** (ISO/IEC JTC 1/SC 7, 2023a) does not include either cost-efficiency or resiliency as characteristics of software systems. Therefore, it was necessary to look for the next Software Engineering Institute reference for an official definition and characteristics to define **resiliency**. This is explained in the following section.

2.1.2 System Resilience as an architectural characteristic

System Resilience (a.k.a system **resiliency**) is the architectural characteristic that is present when the system *"continues to carry out its mission in the face of adversity (i.e., if it provides required capabilities despite excessive stresses that can cause disruptions)"* (Firesmith, 2019b). In other words, *"A system is resilient to the degree to which it rapidly and effectively protects its critical capabilities from harm caused by adverse events and conditions.'* (Firesmith, 2019a). In addition to that definition, Erder (2021) explains clearly that resilience is a different approach to achieving

reliability in which each part of the system is expected to adapt its behavior when a fault occurs, *"tolerating latency, retrying requests, automatically restarting processes, limiting error propagation, and so on"*.

Therefore, it needs to be clarified that for the implementation of high-quality gRPC-based microservices, **resiliency** is crucial through software patterns like timeout, retry, and circuit breaker (Babal, 2024). Microservices are one of the main pillars of MACH architectures, explained in the next section.

2.1.3 MACH Architecture

As mentioned by MACH Architecture (2021a): ***"MACH Architecture** is a set of principles and patterns that define the different building blocks of a new, pluggable, and scalable architecture for creating back-end services and modern user experiences"*.

This architectural approach grants flexible, scalable, and future-proof solutions by using building blocks like LEGO blocks so that the best-of-breed components are utilized. The building blocks of the MACH architecture are the following. **Microservices**, which offer flexibility in deployment, scaling, programming languages, and cleaner test boundaries, have pitfalls such as orchestration costs and maintainability costs. **API-First**, as mentioned, *"is an approach to building software systems where APIs are designed, developed, and deployed as the primary interface for interacting with the system. In other words, the API is built first, and then the underlying system or application is created around it."* (Stevens, 2025), offering a descriptively named contract mapped to a business need. **Cloud-Native** systems leverage all the benefits of the cloud, such as security, agility, and innovation. And **Headless**, which means that *the user experience (what the customer sees) is decoupled from the back-end services, allowing businesses to deliver solutions to a variety of devices in a multi-channel environment in a **cost-effective** and efficient manner.* This offers agility, flexibility, and independent channel delivery of content (MACH Architecture, 2021a).

All of the previous components, together, promote agility, scalability, flexibility, and **swappability**, the last one of which refers to the capacity of the system to have easily replaceable or interchangeable components without disrupting the functionality of the system. All those architectural characteristics represent the value of MACH architectures.

According to the MACH Mindset, it is crucial to have internal collaboration established between DevOps and Business in order to define microservices boundaries and APIs. Moreover, *"by decoupling components, MACH minimizes the impact of system failures"* (Stevens, 2025) to promote resilience. This is also facilitated by cloud technology, which enables teams to update the system without downtime *"and improves overall system reliability"*. (Stevens, 2025). However, in order to properly take advantage of the API, the following considerations should be taken into account: API versioning, backward compatibility of API method's changes, and API evolution as a result of maintenance.

Finally, systems aligned with the MACH architecture might *avoid the large upfront costs of traditional software* by leveraging **cloud native** solutions and **Microservices** (MACH Architecture, 2021a), taking into account the new approach to Total Cost of Ownership (TCO) for Digital Commerce defined by MACH Alliance (2023b), that includes the Total Cost of Change (TCC)

as a new term for measuring architectural changes costs, which is aligned with the *"architect for change"* principle defined by Erder (2021) for continuous architectures.

2.1.3.1 Microservices Architectural Style

Microservices are independently releasable services that are modeled around a business domain or a specific business function, which are accessible to other services via networks (Newman, 2021). All the components that are part of a Microservices architecture form an overall system (Carnell & Huaylupo, 2021; Ford, Richards, Sadalage, & Dehghani, 2021; Newman, 2019).

Microservices need to be implemented by adhering to the following **principles**: highly maintainable and testable; loosely coupled (in both run-time coupling and design-time coupling) (*QCon-Plus 2021: Takeout burritos and minimizing design-time coupling in a microservice architecture*, 2021); independently deployable; organized around business capabilities (modeled around the business domain); owned by a small team; hiding implementation details; isolating failure; and highly observable (Newman, 2015; Richardson, 2020).

In particular, Microservices are deployed in **distributed environments** in which non-functional requirements (quality attributes) like **resiliency**, **reliability, security** should be promoted in order to overcome **network failures** that can cause a malfunction in a specific service (Babal, 2024). Therefore, **resiliency patterns** like **timeout**, **retry**, and **circuit breaker** need to be implemented in the architecture of those systems (Babal, 2024).

This architectural style is needed to write applications that work natively on the cloud (a.k.a cloud native) and has been an increasingly common approach so far because it provides **reliability** (Indrasiri & Suhothayan, 2021). Hence, both developers and architects need to design and develop their solutions to be cloud native by following the Microservices architectural style in order to take advantage of the cloud's benefits.

2.1.3.2 Cloud Native Applications and Patterns

The development of software systems that implement a Microservices architectural style, in which each service *"can run on diversified environments like **public, private, hybrid, or multicloud**"*; *"in an automated, scalable, **resilient**, manageable, and observable way"* is known as **Cloud Native applications** (Indrasiri & Suhothayan, 2021).

The **Cloud Native Computing Foundation (CNCF)** is part of the nonprofit Linux Foundation, and its mission *"is to make cloud native computing ubiquitous"* (Cloud Native Computing Foundation, 2026b). Different software projects participate in this foundation, among which **gRPC** is one of the incubating ones (Cloud Native Computing Foundation, 2026a). This means that the **gRPC** project is widely adopted and stable, but is still building its community.

Finally, in Cloud Native systems, **resiliency** is defined as the *"ability to withstand and recover from disruptions, failures, or unexpected events. It's about preventing failures and ensuring the system can gracefully degrade, recover quickly, and provide some functionality even under stress"* (Schwarz, Moran, & Bachmeier, 2024).

2.1.3.3 API-First and gRPC Inter-Service Interaction Style

*"**API-First** approach ensures that every service is designed for integration and reusability, making it easier to connect systems and share data"* (Stevens, 2025).

Remote Procedure Invocation (a.k.a. Remote Procedure Call (RPC)) is an enterprise integration pattern that *applies the principle of encapsulation to integrate applications.* **Google's Remote Procedure Call technology** , named **gRPC**, is an **inter-service interaction style** that was created by Google in 2015 to enable more efficient communication between **Microservices** and distributed systems by fully utilizing the HTTP protocol in its second version (HTTP/2) (Bass et al., 2021; Hohpe & Woolf, 2004; Indrasiri & Suhothayan, 2021).

HTTP/2, compared to its predecessor (HTTP 1.1), offers the advantages of multiplexing (a single TCP connection between the client and the server), header compression, a binary protocol, and the standard Secure Socket Layer (SSL) recommended by default; all these characteristics not only reduce latency and bandwidth in digital communications but also improve performance in content loading (payload) (Maarek, 2019). Hence, HTTP/2 has become a standard for inter-process (or inter-service or inter-application) communications. On top of HTTP/2, **gRPC** emerged as a technology that grants all those benefits in a single framework. **gRPC** leverages HTTP/2 as a communication backbone for inter-process (interconnected Microservices) communication by using *"a simple language-neutral and platform-neutral **Interface Definition Language (IDL)**"* called **Protocol Buffers v3** (Google, 2022). This interface acts as a single contract that needs to be implemented by both **the client (stub)** and **the server** involved in the communication. Despite its drawbacks (**non-human-readable**, hard to debug, among others), this **IDL** (implemented in **".proto" files**) offers a cross-platform independent API definition that not only promotes bandwidth efficiency by using a binary format that is less CPU-intensive than **JSON** or **XML** formats (commonly used by other architectural styles and protocols like **REST** and **SOAP** because of their human-readability) but also **service interoperability** by delegating the responsibility of its **client (stub)** or **server implementation** to each microservice, regardless of the programming language chosen by the software development teams. Companies like Google, Netflix, and Cisco, among others, use gRPC for their **backend services (server to server communication)** (Adeel, 2025).

On the other hand, **gRPC** provides four different **communication patterns** each of which is used to support different E-Commerce BUCs, as follows (Indrasiri & Kuruppu, 2020). **The Unary RPC pattern**, exemplified by retrieving an order by its ID, involves a client-to-server interaction where the consumer service sends a single request to the order management service and receives the corresponding order. This pattern aligns well with the **BUC1** use case.

Server streaming RPC pattern: addresses scenarios where retrieving multiple related entities is necessary. For instance, when a business requires all orders matching a specific search criterion, the server streams the results to the client incrementally, offering a more efficient approach than retrieving all data at once. This pattern aligns well with the **BUC2** use case.

The Client Streaming RPC pattern is considered suitable for scenarios involving the transmission of a sequence of requests from the client to the server for a single operation, such as updating a set of orders **(BUC3)**.

The Bidirectional Streaming RPC pattern, potentially highly beneficial for **BUC4**, facilitates a continuous two-way communication stream between the client and server. This enables scenarios where the business can simultaneously send a stream of orders and receive processed combined shipments based on the delivery location.

Furthermore, gRPC has implemented different **patterns and tactics** for improving **resiliency**: **timeout pattern** (**deadlines**), **retry pattern**, **circuit breaker pattern**, **error handling**, and **load balancing (client-side and server-side)** (Babal, 2024; Indrasiri & Kuruppu, 2020). Moreover, **security tactics** like **TLS** (Babal, 2024), as well as **authentication mechanisms** like **Token-based authentication with Google** (supporting **OAuth 2.0 tokens** (gRPC Authors, 2024) and **JWT tokens** (Indrasiri & Kuruppu, 2020), which are crucial for the development and cost evaluation of this kind of microservices.

The following table presents the adverse events and conditions for the most important resiliency associated quality attributes for this project, based on Firesmith (2020), and mapped to the gRPC resiliency tactics and patterns presented in (Babal, 2024).

Resiliency Matrix Mapping

Resiliency Associated Quality Attribute	Adverse Events	Adverse Conditions	gRPC Resiliency Tactics and Patterns
Robustness	Input Error Failures	Adverse Environment Faults	Error Handling
Capacity	Load Spikes or Load Related Failures	Excessive Loads	Circuit Breaker
Interoperability	Lost Communications	Degraded Communications	Retry & Timeout

Table 2.1: Resiliency associated quality attributes mapped to gRPC resiliency tactics and patterns.

Finally, the community is currently working on making **gRPC** support HTTP/3 (Brandhorst-Satzkorn, 2019; Newton-King, 2022) by using the QUIC protocol as intended by the Cronet project (Mastrangelo, 2018); this improvement was not considered part of the scope of this thesis, but it is intended for future work.

2.1.4 Amazon Web Services and its Well-Architected Framework

Amazon Web Services (AWS) is a cloud vendor that offers several benefits with the latest cutting-edge technologies in three main service models: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). Some of the most important cloud technology benefits are the following (Amstrong, 2020): scalability and dynamic consumption; vertical scaling; horizontal scaling; elastic load balancing; auto scaling; Geographic Diversity, by using regions, availability zones, and edge locations, grants data proximity, data locality, and privacy regulations; ; among others (Amstrong, 2020). On the other hand, some of the most important **cloud business benefits** are the following (Amstrong, 2020): reducing expenditures and support by offering a **pay**

as you go model, purchasing the infrastructure at the low watermark, and distinguishing costs into hard costs and soft costs; no commitment, which allows businesses to fail fast and innovate by spinning up servers and testing out new business concepts or product enhancements; business agility; disaster recovery (DR) and business continuity (BC); and finally, change to Operational Expenditures (OPEX) by changing accounting from a capital expenditure (CAPEX) to one based on a variable operating cost (Amstrong, 2020).

Regarding cost efficiency, Amazon Web Services (AWS) offers some cloud business benefits for operating services in its infrastructure, with changes to operational expenditures and reduced expenditures and support, among them (Amstrong, 2020). On the other hand, in a migration to AWS, cloud architects should assess connectivity requirements as well as resiliency requirements in order to offer the best experience to customers (Amstrong, 2020).

Finally, AWS offers the possibility of having more predictability of business costs by using tools like **AWS Stock Keeping Unit (SKU), AWS Invoices, AWS Pricing Calculator, AWS Cost Explorer, AWS Budgets, AWS Cost and Usage Reports, AWS Detailed Billing Reports, AWS Trusted Advisor, AWS Consolidated Billing, AWS Credits**, among others (FinOps Foundation Project a Series of LF Projects, 2024b, 2024c).

2.1.5 FinOps, unit economics, cost efficiency and affordability

FinOps's main goal is to offer a new cultural practice to define the way FinOps personas - like **Finance People**, **Engineers**, **Executives and Leadership**, and **Procurement and Sourcing People** - can reduce their communication gaps in order to align technological design decisions and solutions with financial and business goals (**cost efficiency** and **profitability**) in an **inform**, **optimize** , and **operate** cycle (Storment & Fuller, 2023).

From a FinOps perspective, Unit Economics are an effective way to evaluate *"what an organization spends on technology and the value that technology spending creates. Without a way to relate costs to benefits received, it is difficult to understand whether spending is appropriate"* (FinOps Foundation Project a Series of LF Projects, 2025d). Furthermore, it is recommended by the MTF Institute of Management and Finance (2024) that the **average revenue per user (ARPU)** should be significantly greater than the **customer acquisition cost (CAC)** to which for the purpose of this thesis MACH architecture costs belong. Those unit economics helps evaluating cost efficiency.

Cost Efficiency is the ability to maximize profit while minimizing expenditures. Even though it is not defined in (ISO/IEC JTC 1/SC 7, 2023b), to determine **architectural trade-offs** related to it, Firesmith (2009) is proposed as a quality (architectural) characteristic. **Affordability** is defined as *"the degree to which the cost of the system or architectural component lies within budgetary constraints"*. In the Firesmith (2009) proposed quality model, **affordability** includes the following sub quality attributes: acquisition costs, development costs, manufacturing costs, support costs, and retirement costs (Firesmith, 2009).

In order to improve cost efficiency for cloud deployments, in 2016, the term **FinOps** was mentioned for the first time by J.R. Stortment in an Amazon Web Services Conference (Storment & Fuller, 2023), defined as a new cultural complementary model of **DevOps**. FinOps is both a

cultural practice and an operational framework (FinOps Foundation Project a Series of LF Projects, 2024d) *"not just about reducing costs but ensuring that cloud resources are used to support the organization's business goals"* (Trivedi, 2023). FinOps has as its purpose not only to promote the active participation of the financial departments, managed by the **Chief Financial Officers (CFOs)**, in the DevOps cycle but also to build a set of key principles and practices that need to be followed and applied by both Development and Operations regarding the return on investment (ROI) of their architectural and technical decisions, and for the software business to clearly perceive the delivered value either quantified or **estimated** (**forecasted**).

Hence, different industry roles, including IT ones, will require *"poly-skilled employees"* who have a business head, fiscal thinking, and technology acumen. Therefore, this will lead from full-stack engineers to full-business people (Storment & Fuller, 2023)). The following are the common skill sets required to perform FinOps practice (Storment & Fuller, 2023)): **Policy governance:** organization's cost allocation and cloud usage by creating policy documents, standards, Key Performance Indicators (KPIs), Objective Key Results (OKRs) and governance frameworks. **Technical writing:** standards, budget alerts, and documentation of related processes. **Analysis:** look into cloud cost models, report to the different Personas (engineering, finance, business), and detect cost abnormalities. **Engineering:** *"consider the cost of architectural decisions and assist in the automation of billing data, optimizations, reporting of budgets and forecasts, and governance"*. **Automation Engineering:** automate cloud tooling, cloud billing mechanisms (budget alerts and commitments), and recommendations. **Data Engineering:** collect, manage, normalize, and improve data and its reliability and quantity for business analysts. **Training:** self-paced or in person.

A complete engineering solution expected from the FinOps culture includes **capacity, security, business requirements, resiliency, and FinOps cost efficiency** (FinOps Foundation Project a Series of LF Projects, 2024b). It is recommended to have not only the Software Requirements of a solution but also the cost efficiency requirements as part of the other non-functional requirements in the early stages to efficiently design a complete solution (FinOps Foundation Project a Series of LF Projects, 2024b) (FinOps Foundation Project a Series of LF Projects, 2024d). This was the main goal this project had in designing gRPC-based MACH Architectures.

Furthermore, some of the following questions are proposed to think about **FinOps tradeoffs** (FinOps Foundation Project a Series of LF Projects, 2024b): *"How will we allocate costs (by service, by team, by feature)? Could we use different technology that is cheaper? Could we use a different provider with bigger upfront commitments that saves money over time? How efficient do we need to be? How much does this application cost per user? Per engineer? How will we manage anomalies? How will we build the solution so finance reporting needs are met?"* (FinOps Foundation Project a Series of LF Projects, 2024b).

Moreover, *"the primary goal of FinOps persona is to: Accurately **budget**, **forecast**, and **report cloud costs**. This can often be the opposite of the goal of engineers (innovation, delivering value, building efficiency)"* (FinOps Foundation Project a Series of LF Projects, 2024b) (FinOps Foundation Project a Series of LF Projects, 2024d)).

Finally, as part of the relationships between **FinOps Personas and the Engineering Team**: *Leaders use KPIs, NPV, and COGS to set goals and assess value. Engineers must understand and*

be able to defend the decisions and forecasting they make that roll up into these goals (FinOps Foundation Project a Series of LF Projects, 2024b), (FinOps Foundation Project a Series of LF Projects, 2024d).

2.1.6 Total Cost of Ownership (TCO) and Total Cost of Change (TCC)

As Storment says in his FinOps book, *"Understanding TCO is critical to determining the value from your technology purchases and enabling the nirvana state of data-driven decision making"* (Storment & Fuller, 2023).

In MACH Architectures, TCO is defined differently, as the new Digital Commerce approach suggests (MACH Alliance, 2023a). They include a *"better methodology for calculating TCO"* that *"includes evaluation of additional metrics like **Total Cost of Change (TCC)** and Total Spend Productivity (TSP)"*. The first one is defined as *"the cost of making changes to a technology solution over its lifetime."* This was important because *"While TCO is an important metric, it can miss key cost-drivers that have the greatest impact on business outcomes. TCO, inclusive of TCC and TSP, provides a more comprehensive view of the cost of digital commerce solutions and operations, helping budget holders make more informed and impactful decisions about their commerce technology stack"* (MACH Alliance, 2023a).

2.1.7 TCO Cloud Costs Estimation and FinOps savings

As (San Miguel, 2024) says: *"Apart from being able to estimate the potential cost of new projects, we need to be able to analyze the impact of our cost optimization initiatives beforehand"*. Reserved Instances (RIs), for example, *"are ideal for steady-state workloads such as instances that need to be available and running 24/7"* (P. Chung, 2022), but the architecture is tied to an instance type. On the other hand, compute Savings Plans (SP), instead, *"are not tied down to an instance type. AWS will apply the discount rate to any form of compute you use during the term"* (P. Chung, 2022).

Finally, for this project, the TCC and TCO were calculated following these principles, considering the architectural tactics and patterns, as well as the FinOps strategies to optimize the AWS resources that are needed to implement them at a scalable level.

2.2 State of the Art

2.2.1 The Frugal Architect

Dr. Werner Vogels, Chief Technology Officer at Amazon.com, has presented in the AWS re:Invent 2023 (Vogels, 2023a) a new role, **The Frugal Architect** (Vogels, 2023b), as an intent for taking into account the cost per transaction, and per Microservice, Conversion rate/request (value of new features), diminishing returns, to generate a complete costs figure. He proposes to *"think about how to measure that, and make it upfront. Make sure that everybody understands that"*(Vogels, 2023a). Therefore, Architects should design **cost aware architectures** and implement **cost controls** in order to have saving opportunities via Stopping, Rightsizing, Shifting and Reducing cloud resources.

By doing so, architects would be considering **costs as a non-functional requirement** by

following a set of Laws (Vogels, 2023b). Moreover, he explained that **cost optimization** is incremental (continuously optimize) and that the cost to build is less than the cost to operate but software architects need to be conscious about both for **evaluating the cost for an architecture** (Vogels, 2023a, 2023b).

Even though The Frugal Architect Laws (Vogels, 2023b) are concerned about costs as a non-functional requirements for designing cost-aware architectures in the Cloud, there are no specifications about how to implement those Laws in the context of gRPC-based Microservices Architectures.

2.2.2 Amazon Web Services - Cloud Financial Management (CFM)

The **AWS CFM** *"encompasses how you can achieve your business outcomes in the most cost efficient manner, and accelerate economic and business value creation while finding the right balance between agility and control"* (G. Chung, 2023). *The CFM solutions can help transform your business through cost transparency, optimization, forecasting, and control* (G. Chung, 2023). The AWS CFM suggests the following four pillars (G. Chung, 2023): **Measurement and Accountability (see):** strategies like account and tagging, cost reporting and monitoring, **cost showback and chargeback**, and **efficiency** and value **KPIs** are promoted. **Cost optimization (save):** strategies like designing a **cost aware architecture**, match capacity with demand, choose the right pricing model, and identify resource waste are promoted. **Planing and Forecasting (plan): Proof of Concept (POC) based cost estimation** is an activity suggested to be performed. And finally, **Cloud Financial Operations (run):** *"Partnership between Finance and Technology organizations"*.

However, there has not been descriptively and practically specified, up to now, how to adapt this Cloud Financial Management model, its inputs, outputs, and KPIs in order to evaluate the **cost efficiency** of software architectures in general, and for **gRPC-based Microservices on AWS Cloud** in particular.

2.2.3 Cloud FinOps and Kubernetes Optimisation at Scale

Matt Callanan presented a conference about how they manage Cloud FinOps in Enterprise and how they have applied it to optimize costs in Kubernetes deployments at scale (Callanan, 2024). The following are the **Enterprise's FinOps features** (Callanan, 2024): **Savings amortization**; **Free Sharing** (tax and premium fees); **Platform and LM Chargeback**; **Multi-cloud support**; **Organizational Hierarchy Alignment**; **Fix Cloud Provider Data Issues**, including missing instance types and identifying non-taggable resources; **Budgeting and Predictions** based on Machine Learning forecasting; and **Anomaly reporting**.

Even though microservices could be deployed in **Kubernetes** via **EKS**, there has not been a descriptive and practical specification, up to now, on how to adapt this process to evaluate the **cost efficiency** of these kinds of deployments and how they are affected when the microservices to be deployed are **gRPC-based**.

2.2.4 FinOps - Architecting for Cloud

In the context of the Cloud, **FinOps** (Storment & Fuller, 2023) has already defined a set of functional activities, with their respective inputs and outputs, that are expected to be distributively assigned to the FinOps personas—such **as Finance People**, **Engineers**, **Executive and Leadership**, and **Procurement and Sourcing People** (Storment & Fuller, 2023)—for *"designing and modernizing solutions with cost awareness and efficiency to maximize business value while achieving performance, scalability, and operational objectives"*(FinOps Foundation Project a Series of LF Projects, 2024a).

These activities represent a practical approach for designing cost-aware architectures from the FinOps perspective. However, there has not been a descriptive and practical specification, up to now, on how to adapt this process, its inputs, outputs, and KPIs in order to evaluate the **cost efficiency** of **gRPC-based Microservices**.

2.2.5 Amazon Karpenter and Quick Suite

For cloud native, **Karpenter** (Amazon Web Services, 2021) is a tool that helps improving the efficiency and cost of running workloads on Kubernetes clusters by taking into account, scheduling constraints, provisioning nodes, and removing unused pods. These criteria allows Karpenter to reduce infrastructure costs dynamically when needed for containerized cloud native solutions. That makes it partially consider from a software architecture perspective for MACH Architectures in the Cloud-native pillar, but not for gRPC-based nor from software architectural tactics and patterns.

On the other hand, **Amazon Quick Suite** (Amazon Web Services, 2025s) consolidates conversational business intelligence, data dashboards, and multi-agent governance into a single environment that helps overcoming the challenge of gathering *"data across multiple applications—pulling customer details, checking performance metrics, reviewing internal product information, and performing competitive intelligence"* (Amazon Web Services, 2025s) by providing a unified digital workspace, that might help software architects to present dashboards to other departments like the financial one to justify their financial decisions. However it has not been explicitly defined for MACH architectures nor for gRPC-based ones.

2.2.6 Comparison of the state of the art approaches

Unfortunately, there has not been a descriptive and practical specification, up to now, on how to adapt any of the previous approaches in order to evaluate the **cost efficiency** of **gRPC-based Microservices on AWS Cloud**. Therefore, this table presents a comparison between the end-to-end model proposed as the General objective of this project and each of the previous approaches, taking into consideration the following criteria: **Cloud infrastructure centric**: does the approach solve cost efficiency calculations for cloud infrastructures? **Finance centric**: does the approach solve cost efficiency calculations following a financial management approach? **Cloud Architecture centric**: does the approach solve cost efficiency calculations for cloud architectures in general? **Software Architecture centric**: does the approach solve cost efficiency calculations for software architectural styles or patterns? **Application centric**: does the approach solve cost efficiency calculations for software applications? **Platform centric**: does the approach solve cost

efficiency calculations for platforms? **gRPC Interaction Style centric**: does the approach solve cost efficiency calculations for the gRPC interaction style and communication patterns?

State of the Art Framework Comparison

Criterion	Frugal Architect	AWS CFM	FinOps	Enterprise's FinOps	AWS Karpenter & Quick Suite	This Project
Cloud infra. centric	YES	YES	YES	YES	YES	YES
Finance centric	NO	YES	YES	YES	YES	YES
Cloud architecture centric	YES	YES	YES	YES	YES	YES
Software architecture centric	YES	NO	NO	NO INFO	PARTIAL	YES
Application centric	YES	YES	NO	YES	YES	YES
Platform centric	NO	YES	YES	YES	YES	NO
gRPC centric	NO	NO	NO	NO INFO	NO	YES

Table 2.2: Comparison of the state of the art approaches.

2.3 Chapter Summary

This project, in comparison with the other state of the art alternatives, offers a way to include not only the cloud infrastructure and financial side of estimating the cloud efficiency costs of microservices architectures, but also establishes a model to conceive them in the context of MACH architectures whose microservices implement gRPC as an interaction style; an option that has not been contemplated so far, not even by Amazon Web Services or the FinOps Foundation.

Chapter 3

Project Development

"But I do not dare to presume about the applications that I have made of it: I do not ignore the extension of this branch of human knowledge, and I know well the limited sphere of my studies and the weakness of my intellect."

Corrado Avolio, Introduzione allo studio del Dialetto Siciliano 1882
Translated from (Avolio, 2023)

In this section I present my rationale behind my proposed End-to-End model to calculate the cost efficiency of gRPC-based MACH Architectures, starting from the presentation of the conceptual model in which I mapped the different concepts presented in the theoretical framework, the class diagram of the model for the technical tool, and the process software architects should follow to operate the model through the tool.

3.1 Conceptual Design

The conceptual design presented in the following figure synthesizes the mapping of gRPC Interaction Style, AWS Cloud related pillars, and MACH architectural components. This conceptual mapping is key to understand the relationships among the main domains of the end-to-end model considering the elements that involve the design of gRPC-MACH Architectures in the AWS Cloud.

This conceptual mapping places particular emphasis on the **reliability** and **security** pillars, explicitly incorporating **resiliency** through the AWS Resiliency Lifecycle (Amazon Web Services, 2023d; Schwarz et al., 2024). The primary objective of this diagram is to provide a comprehensive visual representation that elucidates how the main MACH Architecture principles for e-commerce systems are mapped with **gRPC** and **AWS Well-Architected Framework** pillars, especially with the **AWS Reliability pillar** (that implies **resiliency** as well) and with the **AWS Security pillar**.

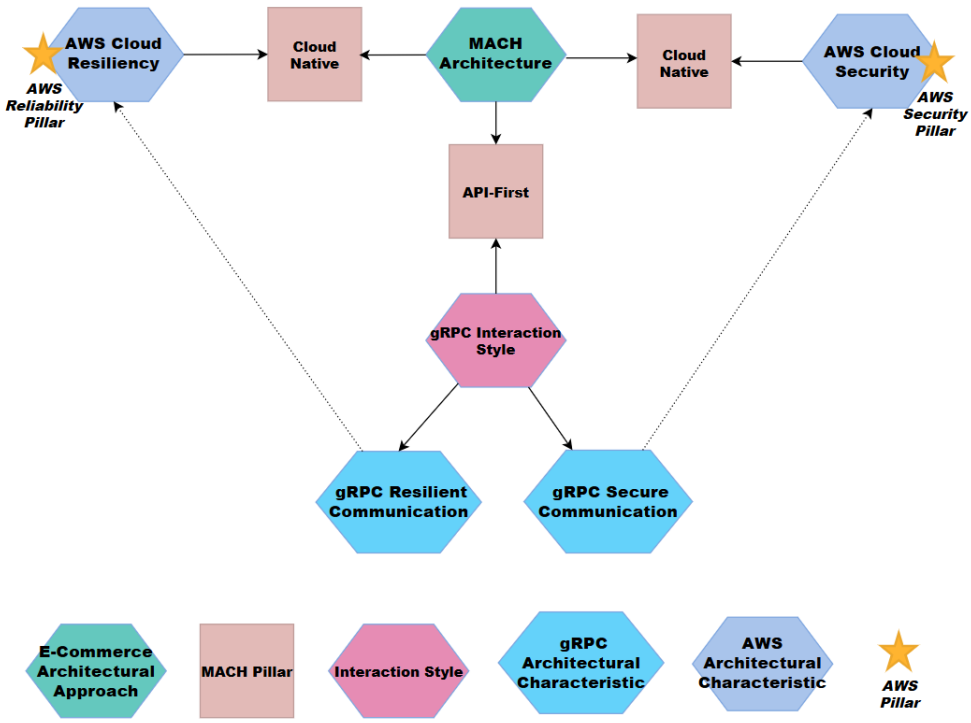


Figure 3.1: Conceptual Mapping - gRPC, MACH Architectures, and AWS Pillars.

As shown in the figure above, it is key to notice that as the gRPC interaction style is intended for server to server communication, the headless component of MACH Architectures was omitted, because no frontend is required for those inter-process communications.

On the other hand, the microservices pillar, being an architectural style, does not have any relationship with any of the AWS Well-Architected Framework pillars, neither with any gRPC communication characteristics. In consequence, that MACH principle was also omitted. Therefore, the rest of the MACH pillars (API-First and Cloud Native) were mapped accordingly, as shown in the following sections.

3.1.1 API-First to gRPC interaction style relationship

Considering that gRPC uses Protocol Buffers to specify the RPC endpoints that will be consumed by another service, and the request and response messages that are part of the API contract, the direction of the arrow pointing to the API-First square corresponds to an instantiation of the API-First pillar through the use of gRPC. This means that gRPC might be replaced by other APIs definitions such as RESTful APIs or GraphQL APIs.

For example, In the following GitHub repository (Chávez, 2026) we can find the **protocol buffer files** that describe the four Business Use Cases under the E-Commerce operational context for a gRPC-based microservice.

1. **BUC1**: **gRPC Unary pattern**: Retrieve orders by using an order ID from the client to the

server. The service offers two endpoints: one for placing orders and another for getting order details, including payment, address, product, and value (money).

2. **BUC2**: **gRPC Server Streaming pattern**: The business needs to retrieve all possible orders that match a search criterion (term or filter). The search criteria are the **order identification** or its **status** (**PENDING, PROCESSING, SHIPPED, DELIVERED, or CANCELLED**). In the complex one, it is offered by the **order identification** or by the **booking identification and user identification**.
3. **BUC3**: **gRPC Client Streaming pattern**: Update a set of orders. Returns a summary of the total updated orders received, the successful ones, and the failed ones.
4. **BUC4**: **gRPC Bi-Directional Streaming pattern**: Send a continuous set of orders and process them into combined shipments based on the delivery date. Returns a stream of combined shipments with the total shipping costs and total items.

Expanding on the first business use case Protocol Buffer (.proto) file content presented in the next two figures, a synchronous gRPC Unary Pattern was defined so that Server A requests order information from Server B utilizing a unique order identifier, The structural anatomy, operational types, and payloads defining the `UnaryOrderService` for BUC1 are structured and parameterized in Table 3.1. This taxonomy maps the service headers, the underlying structural fields, and their implications for serialization.

unaryRPCPattern.proto

```
syntax = "proto3";

package com.ecommerce.order.unary;

option java_package = "com.ecommerce.order.grpc.unary";
option java_multiple_files = true;

import "google/protobuf/timestamp.proto";

enum OrderStatus {
  ORDER_STATUS_UNKNOWN = 0;
  ORDER_STATUS_PENDING = 1;
  ORDER_STATUS_PROCESSING = 2;
  ORDER_STATUS_SHIPPED = 3;
  ORDER_STATUS_DELIVERED = 4;
  ORDER_STATUS_CANCELLED = 5;
}

message Money {
  int64 units = 1;
  int32 nanos = 2;
}

message OrderItem {
  string product_id = 1;
  int32 quantity = 2;
  Money unit_price = 3;
}
```

Figure 3.2: BUC1: Protocol Buffer file part I.

unaryRPCPattern.proto

```
24    message OrderItem {
26      int32 quantity = 2;
27      Money unit_price = 3;
28    }
29
30    message Address {
31      string street = 1;
32      string city = 2;
33      string state = 3;
34      string zip_code = 4;
35      string country = 5;
36    }
37
38    message Order {
39      string order_id = 1;
40      string user_id = 2;
41      repeated OrderItem items = 3;
42      Money total_amount = 4;
43      google.protobuf.Timestamp order_date = 5;
44      OrderStatus status = 6;
45      Address shipping_address = 7;
46    }
47
48    message GetOrderRequest {
49      string order_id = 1;
50    }
51
52    service UnaryOrderService {
53      rpc GetOrderDetails (GetOrderRequest) returns (Order);
54    }
```

Figure 3.3: BUC1: Protocol Buffer file part II.

As mapped in Table 3.1, the communication channel is exposed through a single remote procedure call named `GetOrderDetails` in the contract defined by the Protocol Buffer file. As MACH is API-First software architects define this contract before developing the microservice. This contract (Protocol Buffer file) is important to this model because the calculation of the request and response message sizes are key input values for the estimation of the networking costs that is assumed in the AWS infrastructure when the microservice is deployed in this cloud vendor. To calculate the request and response message sizes the maximum value that each message field might have was considered, so the calculation was closed to the highest possible number of bandwidth the messages might occupy through the HTTP/2 TCP channel, as shown in the code snippet bellow.

Protocol Buffer Interface Map for BUC1 (Unary)

Component Layer	Identifier / Type	Architectural Description & Payload Role
Configuration	`syntax = "proto3"`	Defines wire formatting rules and removes legacy required/optional constraints.
Service Layer	`UnaryOrderService`	Exposes the blocking, point-to-point RPC mechanism for synchronous communication.
RPC Endpoint	`GetOrderDetails`	Maps the execution path: `GetOrderRequest` → `Order`.
Request Payload	`GetOrderRequest`	Low-overhead message containing only the target `string order_id`.
Response Payload	`Order`	Primary data transfer entity carrying domain objects (`items`, `total_amount`, `status`).
Complex Type	`OrderItem`	Represents nested order lines. Declared as a `repeated` field inside the response.
Value Object	`Money`	Highly precise representation of currency splitting units (`int64`) and nanos (`int32`).
Value Object	`Address`	Textual geo-location properties specifying the target customer destination.
Enumeration	`OrderStatus`	State-machine bounds for tracking lifecycle (Pending to Cancelled/Delivered).

Table 3.1: Structural map of the BUC1 unary protocol buffer schema components.

As the microservice needs to be deployed in a cloud vendor, following the MACH Cloud Native principle, the next mapping is between Cloud Native and the AWS Well-Architected pillars.

3.1.2 MACH Cloud Native principle and the AWS Reliability Pillar

The first mapping between **the MACH Cloud Native principles** and the AWS Well-Architected Framework is the one with **the AWS Reliability Pillar**. According to Amazon Web Services (2026m), **cloud resiliency** belongs to the **AWS Reliability Pillar**. This mapping is relevant as the gRPC Interaction Style defines different patterns to promote both reliability and resiliency architectural characteristics, such as client-side load balancing, server-side load balancing, timeout, retry, and circuit breaker for its four communication patterns. Cloud Native is a pillar in which microservices are intended to operate within a cloud vendor, and the services support the microservices operations. The direction of the arrow backwards represents an instantiation of this pillar because this relationship might also be established with other cloud vendors such as Google Cloud Platform (GCP) or Microsoft Azure through their related reliability and resiliency pillars: in GCP, these are defined as **Principles of System Design** (Ghanizada, 2021), and in Microsoft Azure, they are defined as **Azure Cloud Compliance Techniques** (Perkins & Panek, 2021).

As MACH Architectures are API-First, the **gRPC Resilient Communication** patterns and tactics (retry, timeout, and circuit breaker) were included as part of this first relationship between gRPC Resilient Communication and AWS Reliability Pillar. In addition, an error handling model is proposed in the interaction style, considering the status code, status, the error message, and the error details if any error occurs in the gRPC communication between the gRPC endpoints.

The following figures show a deep dive into the different **resiliency patterns and tactics (Retry and Timeout pattern)** for gRPC and their key parameters. The different stimuli under

the retry pattern that can be activated are instant network failures, issues with the availability of services, or resource exhaustion of the service itself. The only one that triggers the use of the Timeout pattern is the alteration of the context under deadlines or cancellation signals.

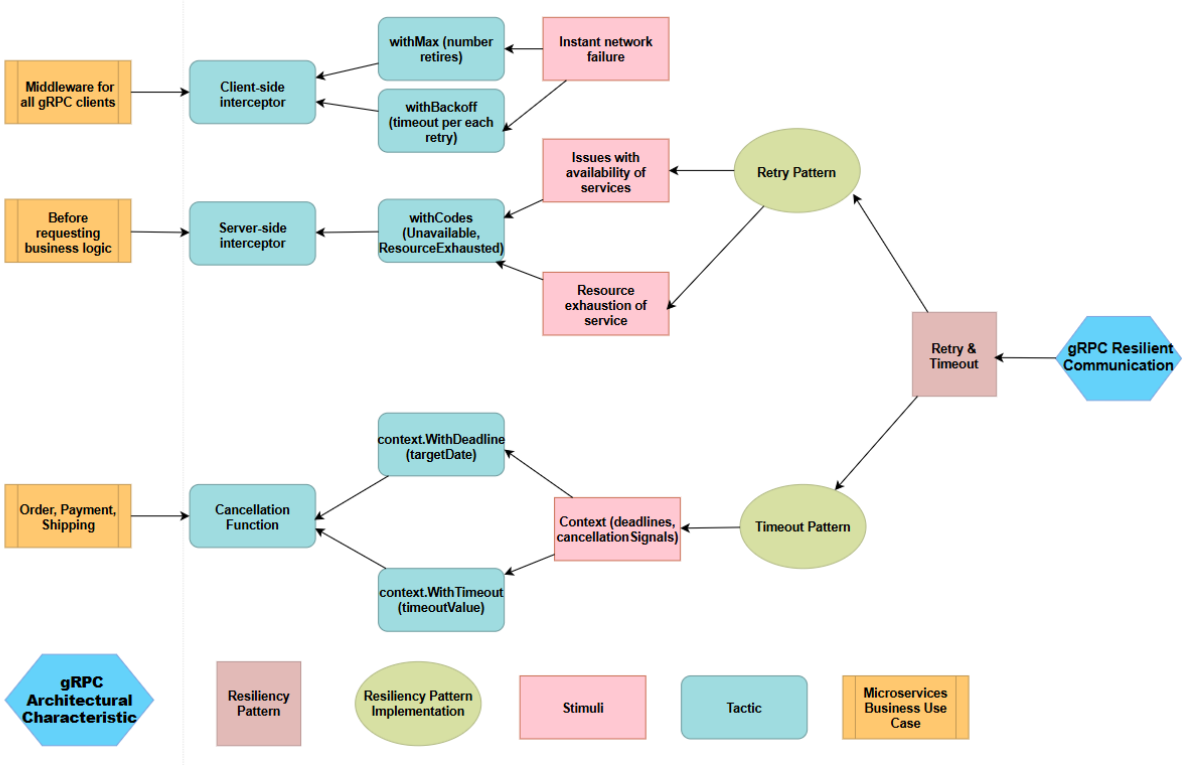


Figure 3.4: Conceptual Mapping - gRPC Resilient Communication - Retry and Timeout Pattern.

Each attribute involved in each pattern and tactic is also a cost factor because, depending on how they are implemented, cost factors related to them can vary significantly, either to be cheaper or to be more expensive than the configuration suggested by default. In consequence, those cost variations impact **affordability**, so each tactic and pattern attributes represent also trade-off grades against other architectural characteristics with **affordability** being the primordial for the **cost-efficiency** calculation.

On the other hand, for the **Circuit Breaker** pattern that is shown in the figure below, the stimuli under its different states are the following: the first one is when the maximum allowed failure ratio is reached, the next one is when the timeout for transitioning is reached, and the last one is when the maximum number of requests allowed is reached.

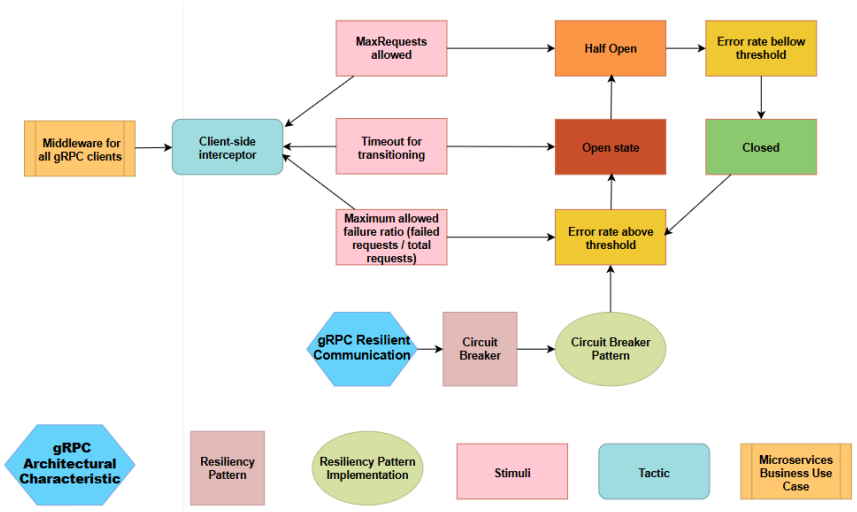


Figure 3.5: Conceptual Mapping - gRPC Resilient Communication - Circuit Breaker Pattern.

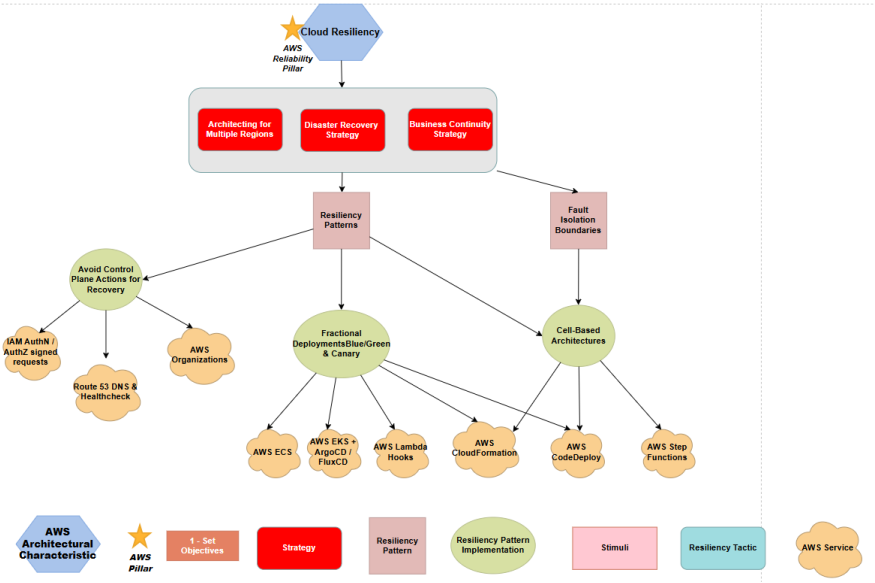


Figure 3.6: Conceptual Mapping - AWS Resiliency Strategies.

Those gRPC Resilient Communication patterns and tactics need to work in a Cloud Native environment following the **Resiliency Lifecycle Framework** (Emer, Richey, Harvey, & Barto, 2023) that is crucial as it implements and validates resilience strategies throughout the application lifecycle (Schwarz et al., 2024). As shown in the next figure, **architecting for multiple regions, for disaster recovery and business continuity** are the key strategies for cloud resiliency.

In order to achieve any level of cloud resiliency some cloud services are important for the development and support of resilient systems in a Cloud Native environment to be considered

mapped as well in the conceptual model as an extension of the AWS pillars, even though Cloud Resiliency is a process that might involve a variety of cloud tactics (such as the AWS Fault Injection Service, chaos engineering, and observability), and few services, such as AWS CloudWatch (Amazon Web Services, 2026m).

Moreover, expanding on the **gRPC Reliability Communication** patterns and tactics, the following is the conceptual mapping representation between those tactics and the AWS Reliability pillar's **REL11-BP02 Fail over to healthy resources**.

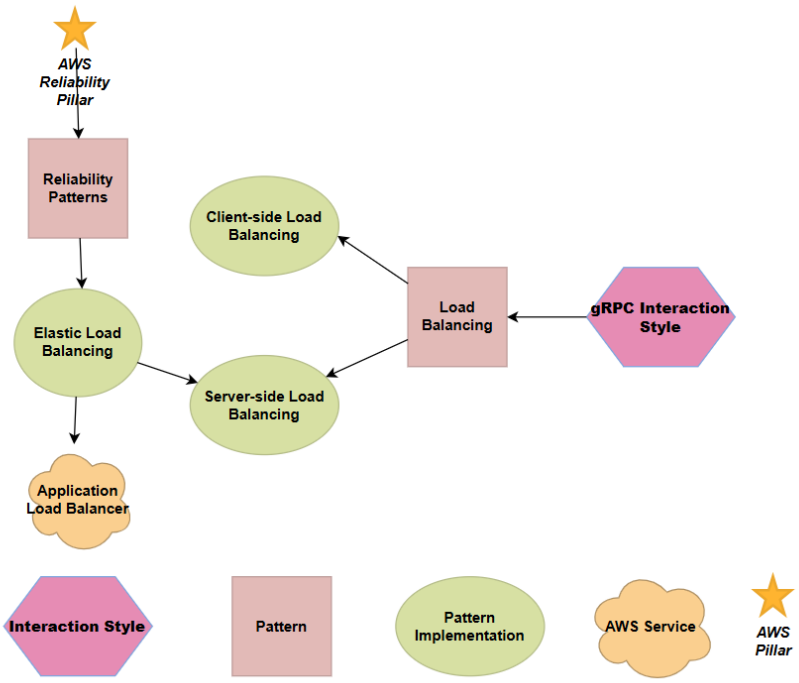


Figure 3.7: Conceptual Mapping - gRPC reliable communication.

It is evident in this case that the Elastic Load Balancing pattern is suggested through the Application Load Balancer service when the gRPC reliable server-side load balancing tactic is thought to be implemented in cloud native environments (Amazon Web Services, 2026k, 2026s).

3.1.2.1 MACH Architecture principles and the AWS Security Pillar

The second mapping link between the **MACH Cloud Native principle** to the AWS Well-Architected Framework is the one with the **AWS Security Pillar**: In contrast to Cloud Resiliency, AWS Cloud Security has its own pillar (Amazon Web Services, 2024d). This conceptual link represented by the backward arrow from AWS Cloud Security and Cloud Native might also be established by other cloud providers such as Google Cloud Platform or Microsoft Azure through their related security pillars as presented above. This **swappability** of MACH Architectures grants having the best of bread solutions without modifying other mappings like the API-First one with gRPC presented in the section above.

As I mapped with resiliency above, expanding on gRPC Secure Communication security patterns and tactics, the following are the conceptual mapping of those tactics that are supported in operation by AWS security cloud services.

In this mapping, software architects identify the AWS Cloud patterns such as Zero Trust and Certificate Authority, the latter being offered in the AWS Certificate Manager service to support TLS credentials for either SSL/TLS or mTLS for mutual verification.

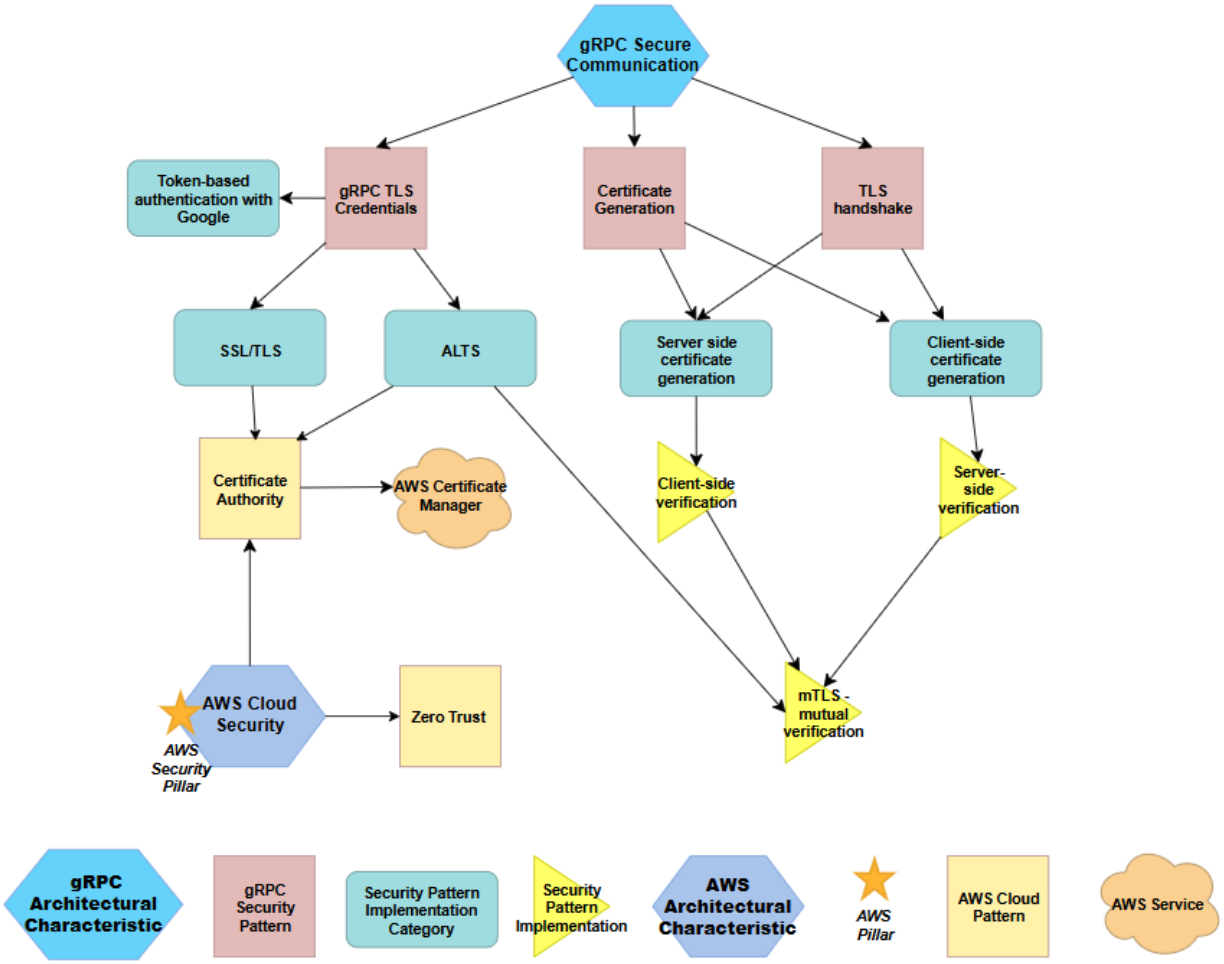


Figure 3.8: Conceptual Mapping - gRPC Security Communication mapping.

On the other hand, for each of the gRPC Secure Communication tactics and patterns there are implementation categories like SSL/TLS, ATLS, Server side certificate generation or CLient-side certificate generation that are related to the main AWS service called AWS Certificate Manager. The advantage of implementing those tactics in AWS is that the AWS Certificate Manager can be used without incurring in additional costs, from the certificate perspective and only the overhead added to the request and response messages of the Protocol Buffer file as a result of enabling either

TLS or OAuth 2.0 with JWT for authentication will be considered for calculating AWS networking costs, when the service is requested by other services in other regions or from another service via internet.

3.1.3 Mapping between MACH Architecture principles, AWS Well-Architected Framework pillars, FinOps Framework, and gRPC Interaction Style

In the appendix **"FinOps+gRPC-Software Architect Tool" (Excel Workbook, Sheet: "4-Cost-Efficient Tool Operation")**, AWS Cloud Resiliency services and AWS Cost optimized architecture factors are directly mapped to FinOps strategies for the Optimize phase according to the specifications presented in San Miguel (2024). In the next table each concept is detailed, considering if each of the key elements of the framework they belong was on scope on this model, partially included (with limitations), or considered for future work. More details are found in the expanded one in the attached Excel file.

Reference Frameworks

Reference	Concept
Introduction to FinOps	FinOps Culture
FinOps Certified Engineer	Engineering Goals
FinOps for Containers	Infrastructure Operations
FinOps FOCUS Analyst	Billing Standardization
MACH TCO	Cost/Benefit Architecture

Table 3.2: Reference Frameworks for the mapping.

As shown in the table above, each reference represents in the model a concept that is useful to specify the conceptual mapping. The concepts and terminology presented in each of those are highlighted in green or red indicating which ones are include in the scope of this project and which ones should be included in future work.

FinOps Framework Structure

FinOps Engineer Goals	FinOps Team Goals	Finance Terminology
Budget	Engineering roadmaps + Financial constraints	Budget
Forecast	N/A	Cost Avoidance
Report	Cost allocation tags	Cost allocation

Table 3.3: FinOps Framework Structure mapping.

Legend: In Scope Future Work

MACH Architectures Framework Structure

MACH Complexity and Costs	TCO as a Metric	APIs as Real-time Cost Control Mechanism
Upfront Costs	Total Cost of Change (TCC)	Cost Management Agility
Total Cost of Ownership (TCO)	Total Spend Productivity (TSP)	N/A
TCO Calculator**	Decommissioned	Data-driven financial decisions
Life-cycle costs	Technical Debt	N/A

Table 3.4: MACH Architectures Framework Structure mapping.

Legend: In Scope Partially Included Future Work

For MACH Architectures, the TCO calculator component whose TCO as a metric definition had been decommissioned from MACH's perspective is partially included due to the fact that in order to perform data-driven financial decisions an initial budgeting and forecast from a calculator is not enough; it is necessary also to have some metrics and data in production environment and reevaluating those calculations as the MACH architecture evolves and operates.

Headless + API-First + gRPC Framework Structure

Interaction Style	ProtocolBuffers .proto file	Operational Context
Headless + API-First + gRPC Interaction Style	Average Size of Request	Estimate the average size in bytes of the request in GB
Headless + API-First + gRPC Interaction Style	Average Size of Response	Estimate the average size in bytes of the response in GB
Headless + API-First + gRPC Interaction Style	Pricing calculation of data transfer per service based on a GB unit	Cloud services calculates the data transfer network costs based on a 1 GB unit

Table 3.5: MACH Headless, API-First, and gRPC Framework Structure mapping.

Legend: In Scope Partially Included Future Work

In terms of the MACH Headless principle, for the cost-efficiency calculation purposes what is necessary to define are the average size of request and of the response as the pricing calculation of **data transfer** per service is based on a GB unit. Therefore, size of requests and responses are calculated in bytes.

For data transfer calls in the Amazon Web Services (2026r), affordability trade-offs can be affected when the traffic is between different cloud regions, within the same cloud region, or out of the internet.

Cloud Native Microservices and FinOps for Containers Structure

Framework Layer	FinOps Engineer Responsibilities	FinOps Engineer Transparency
Cloud Native Microservices + FinOps for Containers	Cost Allocation	All container costs
Cloud Native Microservices + FinOps for Containers	Managing Shared Costs	Cost as a primary requirement
Cloud Native Microservices + FinOps for Containers	Resource Utilization and Efficiency	Mechanisms
Cloud Native Microservices + FinOps for Containers	Managing Commitment - Based Discounts	Engineering and FinOps Team partnership

Table 3.6: Cloud Native Microservices and FinOps for Containers Structure mapping.

Legend: In Scope Future Work

In the context of FinOps for containers (such as Kubernetes), **Managing Shared Costs** is one of the most complex yet critical capabilities. Because containers share underlying infrastructure (nodes, clusters, and networks), attributing costs accurately—rather than just looking at a total bill—is the primary challenge (The Linux Foundation, 2024). This is not included as part of the scope as containers cost are considered for the cost-efficiency calculator as a whole cost factor of the cloud services that enable them such as AWS EKS.

On the other hand, from FinOps perspective the only **unit economics** on scoped are related to the ones reported to CEO and CFOs such as **customer acquisition cost (CA) and average revenue per user (ARPU)** that should be significantly greater than the previous one to consider a MACH architecture cost efficient.

Unit Economics Concepts

Stakeholders	**Description**
Marketing	Advertisement Budget: From Traffic to Leads
Product Manager	Product Metrics: From Leads to Clients
CEO / CFO	Finance Metrics: Customer Acquisition Costs (represented by the costs per user in the unit economics calculated for this model).

Table 3.7: Unit Economics Concepts mapping.

Legend: In Scope Future Work

From the AWS Architecting for requirements (tactics and patterns) perspective, the following table show a classification of those when applied to promote resilient architectures, high-performing architectures, secure architectures, and cost-optimized architectures, the last one of which is directly linked to FinOps strategies such as Reserved Instances and Savings Plans.

Since caching is not needed in gRPC interaction style because it is server to server communication via streaming, high-performing architectures were considered future work.

AWS Architecting for Requirements (Tactics and Patterns)

Concept / Tactic	Description
• **Resilient Architectures**	
AWS Cloud Availability	The percentage of time the application is performing as expected
Hard dependencies	Traditional way of implementing an application even in the AWS cloud. Availability of both EC2 and RDS for an app that interacts with a Database, that has been migrated from on-prem to cloud.
EC2 Autoscaling	Launch Configurations and Templates
Autoscaling Groups	Group of EC2 instances: minimum, maximum and desired
Application Load Balancer Target Group	Balance traffic among EC2 instances in Autoscaling group
Dynamic Scaling Policies	Automatically provision more instances before having overburdened instances: Aggregate CPU utilization, Average request count per target, Average network bytes in, Average network bytes out.
Step Scaling Policies	Based on how aggregate metrics exceeds the thresholds
Target Tracking Policies	Direct value assigned
Scheduled Actions	Have a predictable load pattern on a start date and time and end datetime
Data Backup and Recovery	S3, RDS Instance Snapshot, Multi-AZ RDS, Aurora Replicas (< 16), DynamoDB Global Tables, DynamoDB Point-in-time recovery (35 days to 5min)
Resilient Network	VPC Design Considerations and External Connectivity with Direct Connect
• **High-Performing Architectures**	
Caching	High frequently access data is stored in a faster data layer
Partitioning / Sharding	Horizontal scaling of storage
Compression	Reduce bandwidth usage.
• **Secure Architectures**	
AWS Certificate Manager	SSL/TLS Certificates for encryption
CloudWatch logs	VPC Flow Logs, RDS Logs, Route53 DNS Queries, Lambda, CloudTrail
Amazon GuardDuty	Analyzes and looks for malicious activity in the same service of CloudWatch logs for network traffic
Amazon Inspector	EC2 vulnerabilities
Amazon Detective	Places information in graph databases: VPC Flow Logs, CloudTrail, and GuardDuty
AWS Audit Manager	PCI DSS Standard compliance
AWS Web Application Firewall (WAF)	Monitors HTTP and HTTPS requests
Data encryption in rest	In storage
Data Encryption in transit	AWS Certificate Manager
Macie	PII Compliance in S3 buckets
• **Cost-Optimized Architectures**	
AWS Trusted Advisor	Checks for underutilized or idle resources that might be costing money
Reserved Instances (RI)	Buy the right to use a regular EC2 instance: RI or Convertible RI (change instance)
Savings Plans	1-3 year commitment period either for Compute or EC2
EC2 Spot instances	Temporal EC2 reservation of instances, volatile option
Cost Anomaly Detector	Detects anomalous spend pattern

Table 3.8: AWS Well-Architected Framework - Architecting for Requirements (Tactics and Patterns) mapping.

Legend: In Scope Future Work

Moreover, from the resiliency architectural characteristic defined by Firesmith (2019b), only capacity and interoperability are covered by the gRPC resiliency patterns (Babal, 2024).

SEI Firesmith - System Resilience Quality Attributes Structure

Concept / Tactic	Resiliency Associated Quality Attribute	Adverse Events	Adverse Conditions	gRPC Resiliency Tactics and Patterns
SEI Firesmith - System Resilience	Robustness	Input Error Failures	Adverse Environment Faults	Error Handling
SEI Firesmith - System Resilience	Capacity	Load Spikes or Load Related Failures	Excessive Loads	Circuit Breaker
SEI Firesmith - System Resilience	Interoperability	Lost Communications	Degraded Communications	Retry and Timeout

Table 3.9: SEI Firesmith - System Resilience Quality Attributes Structure mapping.

Legend: In Scope Future Work

All the previous tables include concepts of each framework that were included on the scope of this end to end model or considered for future work. The mapping are not the tables themselves but in how those concepts are mapped in the calculator overview of the **MACH Total Cost of Change (TCC) and Total Cost of Ownership (TCO)**, and the **cost-efficiency (TCC and TCO vs Unit economics)**, presented in the attached Excel file, and whose input variables are presented in the following tables with the conceptual mappings.

gRPC API ProtocolBuffer Size Mapping

Concept / Tactic	gRPC API ProtocolBuffer Size	Value	Description
Pricing calculation of data transfer per service based on a GB unit	IDL ProtocolBuffer Request Size (S_{req})	# bytes	The IDL ProtocolBuffer size in bytes of request (must be <2GiB)
Resource Utilization and Efficiency	IDL ProtocolBuffer Response Size (S_{resp})	# bytes	The IDL ProtocolBuffer size in bytes of response (must be <2GiB)

Table 3.10: gRPC API ProtocolBuffer Size mapping.

Legend: In Scope Future Work

This first mapping is key as the green highlighted concepts that belong to the *Cloud Native Microservices + FinOps for Containers* concepts represent a link between Cloud Native and the gRPC Interaction style considering both request and response sizes as first variables for the calculation of the cost-efficiency.

In the next mapping the gRPC **reliability** tactics are also parameters of the calculator and

this mapping is the same as the one presented in the conceptual diagram above when the ELB in its ALB version of the AWS Well-Architected Framework services is mapped to the server-side load balancing. hence, when calculating the cost-efficiency the costs associated to the AWS ALB service are considered when the software architect decides including that reliability tactic. In contrast, the client-side version of the tactic does not enable ALB.

Reliability Tactics Mapping

Concept / Tactic	Reliability Tactics	Benefits / Description
N/A	Client-side load balancing	Benefits: Balances the emission of gRPC packets to the gRPC server.
Application Load Balancer Target Group	Server-side load balancing	Benefits: Balances the reception of gRPC packets to the gRPC server.

Table 3.11: Reliability Tactics mapping.

Legend: In Scope Future Work

The next mapping is the one between the gRPC **resiliency** tactics mapping with the SEI Firesmith (2019b) resiliency quality attributes as not AWS services are needed to support their implementation. The consequences of selecting those from the cost-efficiency calculation perspective is only linked to the throughput augmentation due to the error % determined as a parameter for the retry tactic. For example if 1000 requests per second is the expected throughput of the gRPC endpoint, and 5% is the expected error rate, therefore 1050 requests per second in total are taking into the consideration for calculating the AWS networking costs.

Resiliency Patterns Structure

Concept / Tactic	Resiliency Patterns	Value / Unit	Benefits & Cons / Description
SEI Firesmith Resiliency - Interoperability	Timeout	# miliseconds	Benefits: Establishes a timeout if a packet has been lost and has not reached the gRPC server.
SEI Firesmith Resiliency - Interoperability	Retry	% error	Benefits: Retries the gRPC packet emission under circumstances of temporal outage of the gRPC service. Cons: Will increment networking costs as more packages are needed to be sent.
SEI Firesmith Resiliency - Capacity	Circuit Breaker	X	Benefits: Balances the reception of gRPC packets to the gRPC server.

Table 3.12: Resiliency Patterns Structure mapping.

Legend: In Scope Future Work

Finally, the last mapping regarding gRPC security patterns is presented in the next table.

Security Tactics Structure

Concept / Tactic	Security Tactics	Benefits / Description
Data Encryption in transit, AWS Certificate Manager	TLS Certificate	Rerefence to calculate the overhead introduced by the TLS standard in each gRPC metwork package.
N/A	OAuth2.0 + JWT Token	Rerefence to calculate the overhead introduced by the Auth 2.0 using JWT Token standards in each gRPC metwork package.

Table 3.13: Security Tactics Structure mapping.

Legend: In Scope Future Work

In this mapping the AWS Certificate Manager is mapped to the TLS Certificate security tactic that might be either TLS or mTLS (mutual in serverA and serverB of a gRPC communication), and Data Encryption in transit is mapped to OAuth 2.0 + JWT Token. Both include overhead to the package transfer in the gRPC communication so both will affect the size of the requests and response messages in the cost-efficiency calculator.

In the last mappings, both Cloud FinOps concepts are mapped with the AWS cloud architecture factors.

Cloud Resiliency - Database and Backup/Recovery Mapping

Concept / Tactic	Component	Detailed Reference / Description	FinOps Pillar / Optimization
Cloud Resiliency - Database and Backup/Recovery	S3	Amazon S3	FinOps Optimize pillar - Storage Optimization
Cloud Resiliency - Database and Backup/Recovery	RDS Instance Snapshot	RDS Instance Snapshot	FinOps Optimize pillar - Snapshot Optimization
Cloud Resiliency - Database and Backup/Recovery	Multi-AZ RDS	Multi-AZ RDS	FinOps Optimize pillar - Database Optimization
Cloud Resiliency - Database and Backup/Recovery	Aurora Replicas	Aurora Replicas (< 16)	FinOps Optimize pillar - Database Optimization
Cloud Resiliency - Database and Backup/Recovery	DynamoDB Global Tables	DynamoDB Global Tables, DynamoDB Point-in-time recovery (35 days to 5min) [Data Backup and Recovery]	FinOps Optimize pillar - Database Optimization

Table 3.14: Cloud Resiliency - Database and Backup/Recovery mapping.

Legend: In Scope Future Work

Cloud Cost Optimized Architecture Factors

Concept / Tactic	Component	Description	FinOps Pillar
N/A	AWS Trusted Advisor	Checks for underutilized or idle resources that might be costing money	Optimize pillar
Managing Commitment - Based Discounts	Reserved Instances (RI)	Buy the right to use a regular EC2 instance: RI or Convertible RI (change instance)	Compute Optimization
	RI	ídem	ídem
	Convertible RI	ídem	ídem
Managing Commitment - Based Discounts	Compute Savings Plans	1-3 year commitment period either for Compute or EC2	Compute Optimization

Table 3.15: Cloud Cost Optimized Architecture Factors mapping.

Legend: In Scope Future Work

Finally, mappings related to microservices and cloud architecture factors and containerized environment factors are included and detailed in the attached Excel file as mentioned above, and in the annexes section at the end of this document.

In the next section the class diagram of the end-to-end model is presented for translating the conceptual mapping to a logical software design for implementing the cost-efficiency calculator using any object-oriented programming language.

3.2 Software Design

Building upon the conceptual design, which synthesizes gRPC communication, AWS Cloud pillars, the FinOps Framework, and MACH architectural components with a particular emphasis on **reliability**, **security**, and **resiliency** architectural characteristics, in this section, I present the detailed Domain Model through a **Unified Modeling Language (UML) class diagram** (Martin, 2003). This diagram serves as a pivotal artifact, providing a comprehensive visual representation that elucidates the static structure of the system and how cloud services, alongside their associated cost implications, support the development, testing, and implementation of gRPC architectural patterns and tactics within MACH solutions adhering to the API-first paradigm.

The diagramming tool that I used was Visual Paradigm (2026) for non commercial use. For the notation, I followed the suggestions presented by Martin (2003).

The following is the notation specification for the class diagram.

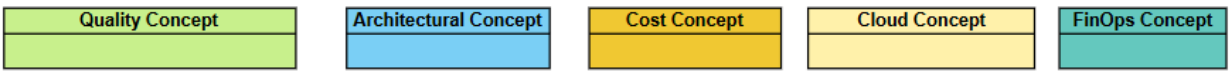


Figure 3.9: End-to-end-Model-Class-Diagram Notation specification.

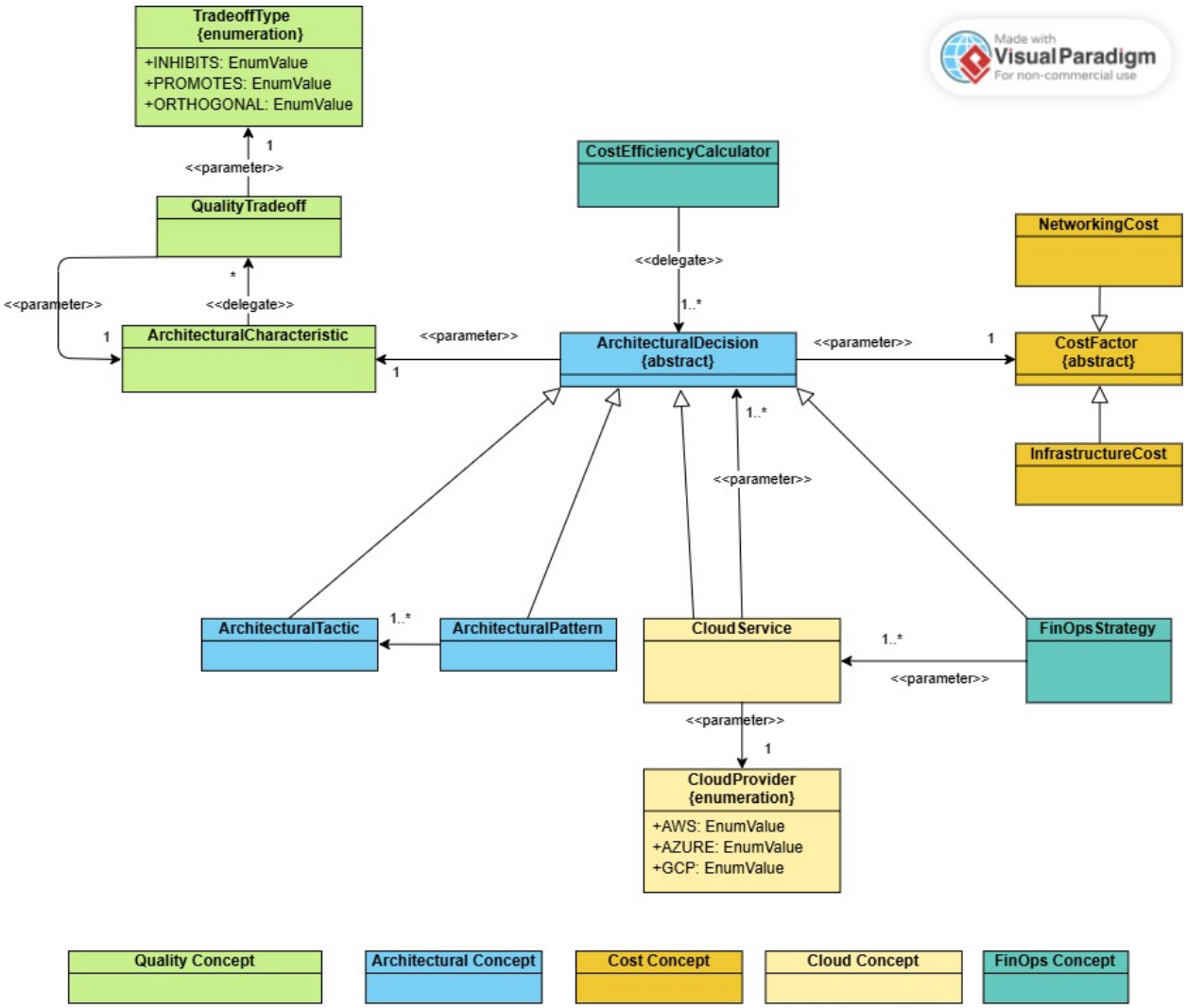


Figure 3.10: End-to-end-Model-Class-Diagram.

3.2.1 Software Architectural Domain

The diagram starts from its center, with the ***[Architectural Decision]*** class that is an abstract representation of any architectural decision a software architect might make in designing a microservice of their MACH Architecture. The architectural decisions available are choosing a business use case (that implies also the Protocol Buffer file of the gRPC endpoint and a gRPC communication pattern to be applied), choosing an ***[ArchitecturalPattern]*** which includes a list of ***[ArchitecturalTactics]***, or choosing an architectural tactic only without being part of an architectural pattern, for example in the case of the gRPC security TLS tactic that is not part of any architectural pattern as defined above. Those classes are in colour blue as they represent the Software Architecture domain of the cost-efficiency calculator.

Each class has a list of attributes and methods aligned with the cost-efficiency calculation purpose. For the software architectural domain the following are the purpose definition of each of those for all its classes.

Class: ArchitecturalDecision

Attribute / Method	Description	Purpose for cost-efficiency calculation
id	Unique identifier for the architectural decision or component	N/A
architectural Characteristic	Defined the promoted architectural characteristic (e.g., reliability, resiliency, security, affordability) affecting the system	N/A
costFactor	Economic criteria or driver associated with the decision	Helps determined if the cost factor is either networking costs or infrastructure costs for the architectural decision

Table 3.16: Class: ArchitecturalDecision.

Class: ArchitecturalPattern (extends ArchitecturalDecision)

Attribute / Method	Description	Purpose for cost-efficiency calculation
name	The descriptive name of the architectural pattern	N/A
architecturalTactics	A list of architectural tactics that compose this pattern	N/A
ArchitecturalPattern (Constructor)	Builder-pattern enabled constructor to initialize pattern attributes and inherited fields	N/A

Table 3.17: Class: ArchitecturalPattern (extends ArchitecturalDecision).

Class: ArchitecturalTactic (extends ArchitecturalDecision)

Attribute / Method	Description	Purpose for cost-efficiency calculation
name	The descriptive name of the architectural tactic	N/A
ArchitecturalTactic (Constructor)	Builder-pattern enabled constructor to initialize tactic attributes and inherited fields	N/A

Table 3.18: Class: ArchitecturalTactic (extends ArchitecturalDecision).

3.2.2 Quality domain

The quality domain is the one represented with colour green. Two classes represent the ***[QualityTradeoff]*** between the ***[ArchitecturalCharacteristic]*** promoted by an architectural decision and other architectural characteristics that might be either promoted or inhibited. The most important trade-off for each architectural decision is the ones established with affordability because it is the architectural characteristic more proximal to cost-efficiency from a FinOps budgeting perspective. For the quality domain the following are the purpose definition of each of those for all its classes.

Class: ArchitecturalCharacteristic

Attribute / Method	Description	Purpose for cost-efficiency calculation
name	The descriptive name of the architectural characteristic	N/A
qualityTradeoffs	A list of tradeoffs associated with this characteristic	The quality trade-offs of selecting other architectural characteristics are established against affordability before establishing it with cost-efficiency. This trade-off happens each time a software architect includes any of the gRPC tactics or any of the cloud services

Table 3.19: Class: ArchitecturalCharacteristic.

There are different architectural characteristics that can be included in their catalogue. However, the most important trade-offs to analyse are the ones established between reliability, resiliency, and security with affordability. Those trade-offs happen each time a software architect includes any of the gRPC tactics or any of the cloud services. Those trade-offs are the key ones for this end-to-end model as they represent the initial approach of calculating the Total Cost of Change, with change meaning choosing or discarding a reliability, resiliency or security tactic, a cloud service or a FinOps strategy.

Enum: ArchitecturalCharacteristics

Constant	Description	Purpose for cost-efficiency calculation
RELIABILITY	Ability to perform required functions under stated conditions	N/A
RESILIENCY	Ability to recover from adverse events and conditions	N/A
SECURITY	Protection of information and system resources	N/A
AFFORDABILITY	Ability to maintain costs within budget constraints	The quality trade-offs of selecting other architectural characteristics are established against affordability before establishing it with cost-efficiency.

Table 3.20: Enum: ArchitecturalCharacteristics.

Class: QualityTradeoff

Attribute / Method	Description	Purpose for cost-efficiency calculation
architecturalCharacteristic	The characteristic impacted by this tradeoff	N/A
tradeoffType	The nature of the relationship (Promotes, Inhibits, Orthogonal)	It is useful when the tradeoffType promotes or inhibites affordability. When it is inhibit that means that the selection of an architectural decision leaves the software architect out of budget to build the MACH microservice

Table 3.21: Class: QualityTradeoff.

Enum: TradeoffType

Constant	Description	Purpose for cost-efficiency calculation
PROMOTES	The decision positively influences the characteristic	It is useful when the tradeoffType promotes affordability. When it is promoted that means that the selection of an architectural decision helps the software architect reduce costs and in consequence to have more budget to build the MACH microservice. This usually happens when a FinOps strategy for reserved instances or commitment discounts are chosen by the architect
INHIBITS	The decision negatively influences the characteristic	It is useful when the tradeoffType inhibites affordability. When it is inhibit that means that the selection of an architectural decision leaves the software architect out of budget to build the MACH microservice.
ORTHOGONAL	The decision has no significant influence on the characteristic	It is useful when the tradeoffType is with affordability. When it is orthogonal that means that leaves the software architect in budget to build the MACH microservice

Table 3.22: Enum: TradeoffType.

3.2.3 Cloud Domain

Another kind of architectural decisions that might be made is choosing cloud services to support the software architectural ones. Each ***[CloudService]*** belongs to a ***[CloudProvider]*** like AWS, GCP or Azure. As a differential characteristic, a single cloud service can support the implementation in a cloud provider, for multiple architectural decisions, or at least one. For example, the AWS Certificate Manager, as shown in the conceptual diagram supports the TLS and mTLS gRPC security tactics. These classes are in light yellow as they belong to the cloud domain of the cost-efficiency calculator.

The possibility of choosing among different cloud providers is the starting quote of having the best-of-bread components for a MACH Architecture that is reflected in the class diagram for the cloud domain. Enabling GCP and Azure or even Oracle Cloud means the software architect in the future will count on a calculation that adapts to different cloud vendors or multi-cloud solutions. This is essential for MACH Architectures in which the best-of-bread components need to be integrated, independently of their vendor.

The extensibility provided in this cloud domain in the class diagram deals with the complexity that sometimes software architects have in choosing the right services or cloud providers when designing software architectures that need to be cloud native. In the class diagram this is facilitated as a key aggregated value that helps the software architect to be a Frugal one. For the cloud domain

the following are the purpose definition of each of those for all its classes.

Class: CloudService (extends ArchitecturalDecision)

Attribute / Method	Description	Purpose for cost-efficiency calculation
cloudProvider	The specific cloud vendor (e.g., AWS, GCP, Azure)	Each cloud provider defines its service pricing APIs and FinOps strategies like reserved instances and commitment discounts.
name	The descriptive name of the cloud service	N/A
architecturalCharacteristic	Defined traits associated with the service	N/A
supportedArchitectural Tactic	The architectural tactic enabled by this cloud service	N/A
infrastructureCost	The cost object representing service consumption	Each cloud service has different pricing schemas when operating them on cloud and their prices are the key values to add them as TCC when those are considered by a software architect
supportedArchitectural Decisions	List of other decisions supported by this service	N/A
CloudService (Constructor)	Builder-pattern enabled constructor to initialize fields and superclass state	N/A

Table 3.23: Class: CloudService (extends ArchitecturalDecision).

Enum: CloudProvider

Constant	Purpose for cost-efficiency calculation
AWS (Amazon Web Services provider)	For AWS the following are the available pricing services and APIs: the **bcm-pricing-calculator** (Amazon Web Services, 2026), the **AWS Price List Query API** and the **AWS Price List Bulk API**, the **AWS Calculator**, and the **AWS Migration Evaluator** (San Miguel, 2024)
GCP (Google Cloud Platform provider)	For GCP the following are the available pricing services and APIs: the **GCP Cloud Pricing Calculator**, the **GCP TCO Assessment**, the GCP Migration Center, the **GCP Cloud Billing Catalog API**, and the **GCP Custom Pricing API** (San Miguel, 2024)
AZURE (Microsoft Azure provider)	For Azure the following are the available pricing services and APIs: the **Azure Calculator**, the **Azure TCO Calculator**, the **Azure Retail Prices REST API**, and the **Azure Price Sheet API** (San Miguel, 2024)

Table 3.24: Enum: CloudProvider.

Note: for the purpose of this thesis, the **AWS region** in which the pricing requests to the APIs are performed is the **us-east-1**, in the **US East (N. Virginia) location**. This location has been chosen as it has the region that has the most complete catalogue of AWS services among the regions. However, it is necessary to clarify that changes in region might generate changes in services pricing (Amazon Web Services, 2026f).

3.2.4 FinOps cost-efficiency domain

The last kind of architectural decisions that a software architect can make is related to the different ***[FinOpsStrategy]*** applied to one or more cloud services. Either through reserved instances or commitment discounts, or spot instances, applied to cloud services the software architect is able to promote affordability, and in consequence promote cost-efficiency as well, by reducing costs.

In addition, the class that represents the ***[CostEfficiencyCalculator]*** I decided it to be part of this FinOps cost-efficiency domain because this class is responsible for calculating the unit economics by microservice based on the architectural decisions made by the software architect.

For the FinOps cost-efficiency domain the following are the purpose definition of each of those for all its classes.

Class: FinOpsStrategy (extends ArchitecturalDecision)

Attribute / Method	Description	Purpose for cost-efficiency calculation
name	The descriptive name of the FinOps strategy	N/A
cloudServices	Relationship indicating the strategy is applied to at least one or more cloud services	It is important as each FinOps strategy applied to one or more cloud services might represent a relevant promotion of affordability
FinOpsStrategy (Constructor)	Builder-pattern enabled constructor to initialize strategy attributes, cloud service associations, and inherited fields	N/A

Table 3.25: Class: FinOpsStrategy (extends ArchitecturalDecision).

Class: CostEfficiencyCalculator

Attribute / Method	Description	Purpose for cost-efficiency calculation
architecturalDecisions	Relationship to architectural decisions being analyzed	By analyzing each architectural decision and its quality trade-offs with affordability the calculator will perform accordingly the unit economics that enable the software architect in determining the cost-efficiency of a microservice
calculateUnitEconomics()	Computes and returns UnitEconomicsResponse: Includes TCO (Monthly/Annual, Egress, Cloud Infra, FinOps Savings), Unit Costs (Per Request, Per User/Month/Day, Total Requests), and ROI/Profitability metrics (Revenue data, ARPU, ROI%, Net Profit, Break-even, Revenue/Dollar, Net Margin)	The key for determining if a microservice or a MACH Architecture is cost efficient is by comparing the value of the Average Revenue Per User (ARPU) that must be significantly greater than the Customer Acquisition Cost (CAC), represented by the Cost per User.

Table 3.26: Class: CostEfficiencyCalculator.

3.3 Operationalization

Having defined the essential architectural concepts and guiding principles for evaluating cost efficiency in gRPC-based MACH architectures, I decided to design a process that describes how these architectural concepts are systematically applied in practice, by using the Business Process Model and Notation (BPMN) and the Bizagi Modeler tool (Vanner, 2024). The operative process is presented in the following diagram.

In the process, the software architect starts choosing an **architectural driver** (architectural characteristic that is key for their MACH Architecture like resiliency, reliability or security). This is because software architects usually start by defining those drivers as a result of having understood accordingly the business quality priorities over a list of specifications and requirements.

As software architects, once they know what are the architectural drivers of a system, then, for the selected attribute, the software architect needs to identify suitable architectural tactics (e.g., retry pattern, circuit breaker, TLS handshake) to be chosen, and select the most convenient based on the architectural driver. Once at least one is chosen, the calculator calculates the TCC for the selected tactic. This is important in order to make the affordability visible to the software architect "just-in time".

Then, the software architect needs to map each tactic to specific cloud services offered by providers such as AWS, Azure, or GCP. This is needed as not only the architectural drivers are developed using a software pattern and tactic but also cloud services like ALB, when the system

is cloud native. For example, AWS ELB /ALB should be shown too be selected if server-side load balancing reliability tactic is required to be implemented in the gRPC-based microservice.

In consequence, the calculator calculates the TCC for the selected cloud service but differentiating networking costs of infrastructure costs. This differentiation is crucial for letting know the software architect abou the costs related to the gRPC message payloads and throughput from the cloud services related ones. With this differentiation, the software architect can make decisions either on the Protocol Buffer contract to reduce message payloads or overhead to lower networking costs, or apply FinOps strategies when dealing with high cost cloud services.

Once the cloud services are selected, the software architect would need to select among FinOps strategies to optimize costs, such as reserved instances, spot instances or commitment discounts. Those will adjust cloud service configurations in accordance with the chosen optimization strategies, either promoting, inhibiting or orthogonalizing the affordability.

Therefore, the calculator needs to recalculate the costs based on the optimized configuration, generating a detailed financial forecast for a month and a year, using the unit economics of the FinOps domain.

Then the cost-efficiency calculator act over all the architectural decisions to calculate the networking costs and infrastructure costs along with the unit economics by using the different official cloud pricing calculators. This information is key to be provided to the software architect to finally identify that not only the gRPC-based microservice is affordable (from affordability) and cost-efficient if the ARPU is significantly greater than the CAC per microservice or per request.

Finally, the software architect needs to check if all relevant architectural drivers have been addressed or if another one is pending to be included. If so, they repeat the process until all architectural drivers have been covered for the MACH Architecture microservice.

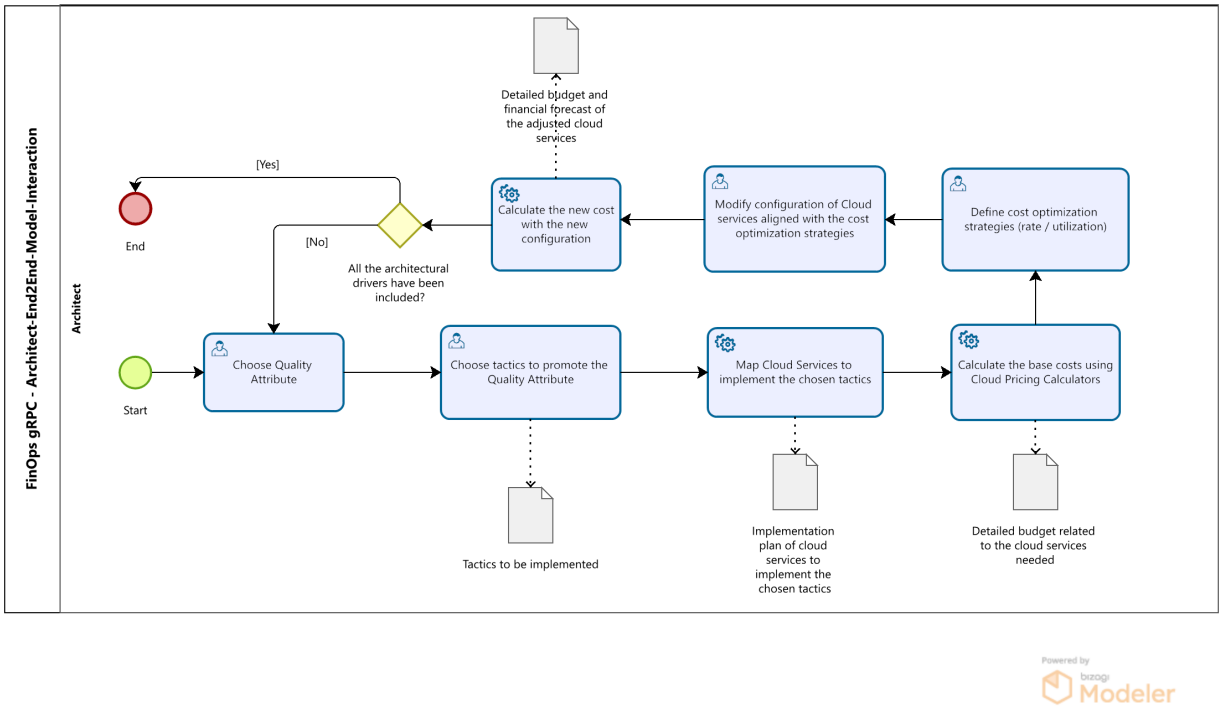


Figure 3.11: End-to-end model interaction process.

This operational process is iterative because it ensures that architectural decisions are both technically aligned with business needs, cost-aware, and financially justified. Therefore the software architect needs to be careful about how each architectural tactic and pattern promotes or inhibits affordability each time. That solves the necessity for MACH Architectures of have a clear view of their TCC. At the end the TCO estimated is just a reflection of the group of choices made in each change.

Moreover, by integrating FinOps practices, cloud service mapping, and iterative cost optimization, we, as software architects, can deliver solutions that balance quality attributes with cost-efficiency, providing clear justification for FinOps Personas (FinOps Foundation Project a Series of LF Projects, 2025i)

Finally, in the annexes section, a mapping between the operationalization to the FinOps framework is done to conceptually identify the FinOps principles that are linked.

Chapter 4

Implementation

"Building the product right and building the right product are two different things."

Gojko Adzic, 2011
(Adzic, 2011)

The **gRPC MACH Architecture TCO Networking Costs Calculator** was designed for **software architects acting as FinOps engineers**, bridging the gap between information technology and finance by translating architectural decisions into measurable financial impacts on AWS infrastructure.

The tool follows a **five-phase wizard** that mirrors the architect's decision-making journey: from service identity definition, through tactical selection, cloud infrastructure configuration, and finally to a full Total Cost of Ownership (TCO) report and multi-service portfolio analysis. Each phase is backed by a modular **Thymeleaf** frontend communicating with a **Spring Boot** backend that calls the **AWS Price List API** (both Query API and Bulk API via AWS SDK v2) to obtain live, service-specific pricing data, specifically from the **us-east-1 region**, the **US East (N. Virginia) location** (Amazon Web Services, 2026f).

As part of the software requirements analysis phase, the next section presents the Specification by Example approach that was used to specify them and to test them accordingly.

4.1 Software Requirements for the Cost-efficiency calculator

Following the **Specification by Example** methodology (Adzic, 2011), all functional requirements are expressed as executable *Given-When-Then* scenarios that simultaneously serve as living documentation and the acceptance-test baseline for each phase of the tool. The scenarios were structured according to the five phases of the wizard and are presented as **user stories** that can be consumed by both business stakeholders and the development team.

Adzic (2011) defines the **Specification by Example** (SBE) as a collaborative method for specifying requirements and tests that uses seven patterns to effectively produce living, reliable, and testable documentation. It is very useful for reducing manual testing and rework, ensuring that delivery teams and business stakeholders are confident that the software being built is fit for its purpose.

The term SBE is directly related to the most recent ones: **Acceptance Test-Driven Development (ATDD)** (Beck, 2002) and Behavior-Driven Development (BDD) (Smart & Molak,

2023), the latter created by Daniel Terhorst-North (a.k.a. Dan North). Adzic (2011) prefers to use SBE instead of ATDD or BDD, as it provides a general conceptualization of the method that accomplishes the purpose of closing the gaps between the business and the technical parts of a software solution by promoting their collaboration for requirements specifications.

SBE solves one of the most difficult challenges in software development projects: to work as a base to sign-offs by providing living documentation as an artifact that validates executable specifications, which can also be kept in version-control systems, and by providing stability in the business rules more than in the technology that implements them.

The methodology leverages Specification by Example, using Cucumber's Given-When-Then syntax (Wynne, 2015) to define features and scenarios for each activity in the architect's process. Each feature corresponds to a distinct activity, such as selecting quality attributes - including reliability, resiliency, and security tactics - and is illustrated with user stories and scenario outlines that reflect real-world architectural choices. The scenarios are aligned with the class diagram, ensuring consistency between requirements, architecture, and implementation artifacts.

By grounding use cases in both functional and architecturally significant requirements, this approach enables architects to systematically address quality attributes such as reliability (e.g., server-side and client-side load balancing), resiliency (e.g., timeout, retry, and circuit breaker patterns), and security (e.g., TLS handshake, certificate generation, gRPC TLS credentials) (Babal, 2024). Furthermore, the integration of FinOps operational tactics and cloud service considerations ensures that cost optimization and operational efficiency are embedded from the outset. This structured specification process not only clarifies the **software architect's** role in requirements engineering but also provides a robust foundation for validating that the architecture supports both functional and non-functional goals throughout the system lifecycle.

Therefore, the following is the result of the mapping of the User Stories, the features aligned with them, and the different Given-When-Then scenarios (Adzic, 2011) (Adzic, 2020). **Note**: as recommended in (Adzic, 2011), the **user stories** are formulated first to clearly delineate the responsibilities of the Business User—represented in this context by the **software architect**—which corresponds to the **"As a ..."** and **"So that ..."** statements—and the specific requirements expressed in the **"I want..."** statements, which the developer (in this case, the author of this thesis) must satisfy through the implementation of the **right software solution** (Adzic, 2011).

Agile Requirements Mapping Strategy

User Story Statement	Stakeholder or Business User	Tool's Developer
As a user ...	✓(Software Architect).	-
So that ...	✓(Something valuable for the Software Architect: output + goal).	-
I want ...	-	✓(System function).

Table 4.1: Structure of a User Story with the Software Architect being the Business User, and the Developer the responsible to develop the system function (Adzic, 2011).

Activity 1: Choose Quality Attribute - User Story: Software Architectural Characteristic/Driver/Quality Attribute Selection

As a Software Architect
So that I can design a MACH Architecture that meets business-context-driven non-functional requirements
I want to select appropriate quality attributes

Scenario Outline: Quality attribute and architectural tactics mapping
Given a project with ***<business_requirement>***
When I select ***<architectural_characteristic>*** as an architectural driver
Then the system should suggest ***<recommended_tactics>*** as potential implementation options

Architectural Feature Examples

business_requirement	**archit_charac**	**recommended_tactics**
Failure recovery needs	Resiliency	Timeout, Circuit Breaker, Retry
Correct operation over time needs	Reliability	Client-side Load Balancing, Server-side Load Balancing
Data protection needs	Security	TLS (One-way), mTLS (Mutual TLS), OAuth 2.0 + JWT

Table 4.2: User Story: Software Architectural Characteristic/Driver/Quality Attribute Selection - Feature Examples

The sequence diagram of this activity can be reviewed in the annex section.

Activity 2: Choose Architectural Tactics or Patterns to Promote the Quality Attribute - User Story: Architecture Tactics and Patterns Selection

Feature: Architectural Tactics and Patterns Selection
As a Software Architect
So that I can implement technical strategies addressing business needs
I want to select specific tactics or patterns for my chosen quality attributes

Scenario Outline: Tactic selection to promote quality attributes
Given I have selected *"<architectural_characteristic>"* as the primary focus.
When I choose *"<specific_tactic>"* as the implementation approach.
Then the system should identify *"<impacted_attr>"* with *"<impact_type>"*

Architectural Tactics & Patterns Mapping

arch_charac	specific_tactic	impacted_attr	impact_type
Resiliency	Timeout	RESILIENCY	PROMOTES
Resiliency	Retry	RESILIENCY	PROMOTES
Resiliency	Circuit Breaker	RESILIENCY	PROMOTES
Reliability	Client-side Load Balancing	RELIABILITY	PROMOTES
Reliability	Server-side Load Balancing	RELIABILITY	PROMOTES
Security	TLS (One-way)	SECURITY	PROMOTES
Security	mTLS (Mutual TLS)	SECURITY	PROMOTES
Security	OAuth 2.0 + JWT	SECURITY	PROMOTES
Resiliency	Timeout	AFFORDABILITY	ORTHOGONAL
Resiliency	Retry	AFFORDABILITY	INHIBITS
Resiliency	Circuit Breaker	AFFORDABILITY	ORTHOGONAL
Reliability	Client-side Load Balancing	AFFORDABILITY	ORTHOGONAL
Reliability	Server-side Load Balancing	AFFORDABILITY	INHIBITS
Security	TLS (One-way)	AFFORDABILITY	INHIBITS
Security	mTLS (Mutual TLS)	AFFORDABILITY	INHIBITS
Security	OAuth 2.0 + JWT	AFFORDABILITY	INHIBITS

Table 4.3: User Story: Architecture Tactics and Patterns Selection - Feature Examples

For the first two activities, the software architect can select different architectural characteristics in the tool and check their tactics and patterns to be implemented in the TCC and TCO cost-efficiency estimation. The following is an example with the Reliability patterns and tactics. In addition, in this case, as the server-side load balancing reliability tactic was selected, the AWS application load balancer was also automatically selected.

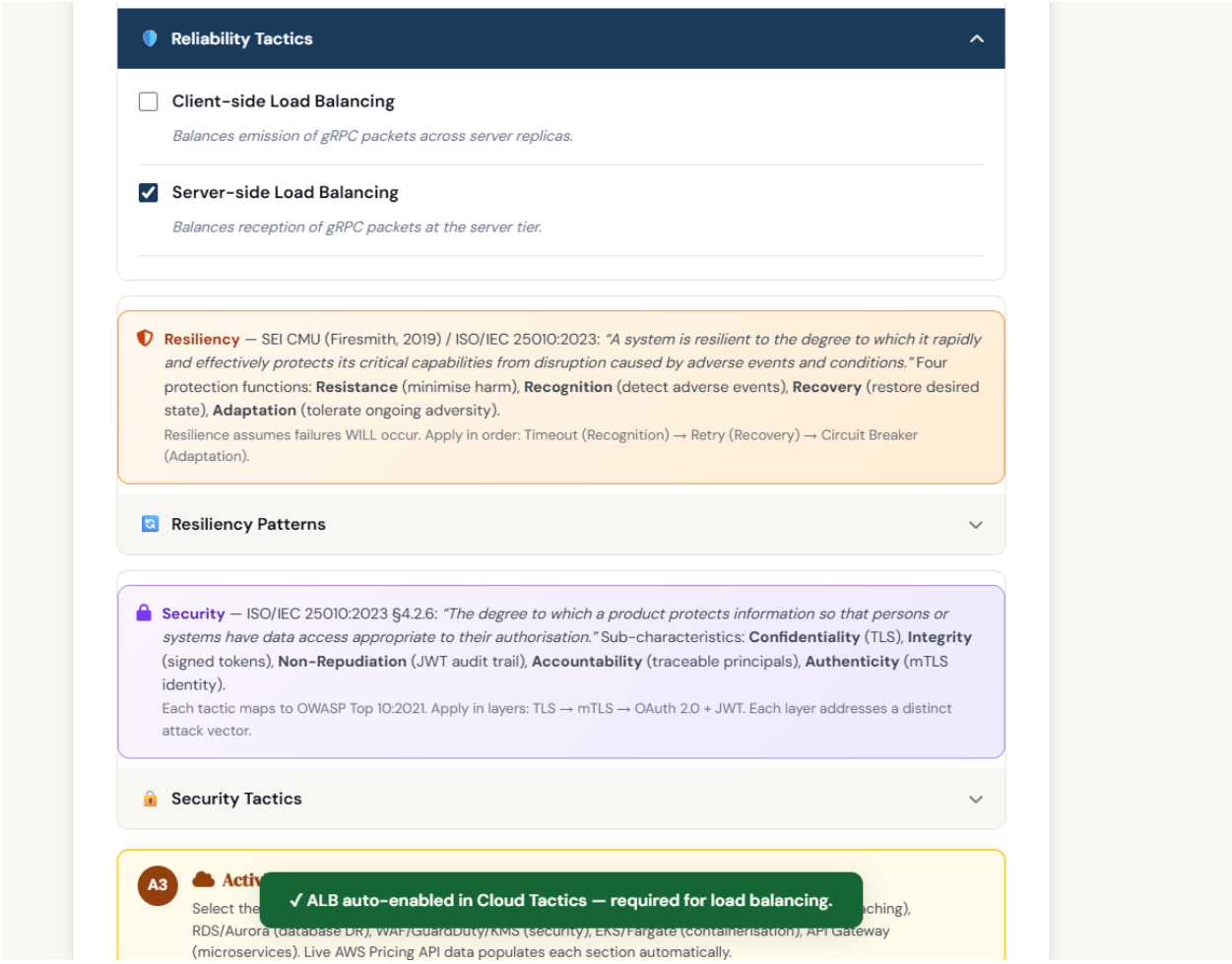


Figure 4.1: End-to-end model - Activity 1: Choose Architectural Characteristic & Activity 2: Choose Architectural Tactics and Patterns.

In the previous figure it is evident as well the relationships between architectural characteristics and the tactics and patterns offered as options to be selected and parametrised by the software architect using the cost-efficiency calculator. This is important as it represents the navigability offered by the quality and architecture domains of the software diagram of the end-to-end model.

Activity 3: Map Cloud Services to Implement the Chosen Tactics - User Story: Cloud Service Mapping

Feature: Cloud Service Mapping
As a Software Architect
So that I can implement a cost-aware architecture using concrete cloud provider offerings
I want to map selected tactics to specific cloud services

Background:
Given the architect is authenticated on the TCO Calculator application
And the architect is configuring tactics in the Cloud Tactics & Patterns panel

Scenario Outline: Mapping architectural patterns and tactics to AWS cloud infrastructure
Given I have selected pattern or tactic *"<pattern_or_tactic>"* for implementation that promotes *"<architectural_characteristic>"*
When I specify *"<cloud_provider>"* as my deployment target
Then the system should map the configuration to specific services *"<specific_services>"*

Cloud Service Mapping Strategy

arch_charac	pattern_or_tactic	cloud_prov	specific_services
RELIABILITY	Server-side Load Balancing	AWS	Elastic Load Balancer - ALB Layer 7
SECURITY	TLS (One-way)	AWS	Amazon Inspector, Certificate Manager, AWS WAF, Amazon CloudWatch, AWS Audit Manager, AWS KMS
SECURITY	mTLS (Mutual TLS)	AWS	Amazon Inspector, Certificate Manager, AWS WAF, Amazon CloudWatch, AWS Audit Manager, AWS KMS
SECURITY	OAuth 2.0 + JWT	AWS	Amazon GuardDuty, AWS CloudTrail, Amazon Macie, Amazon CloudWatch, AWS Audit Manager

Table 4.4: User Story: Cloud Service Mapping - Feature Examples

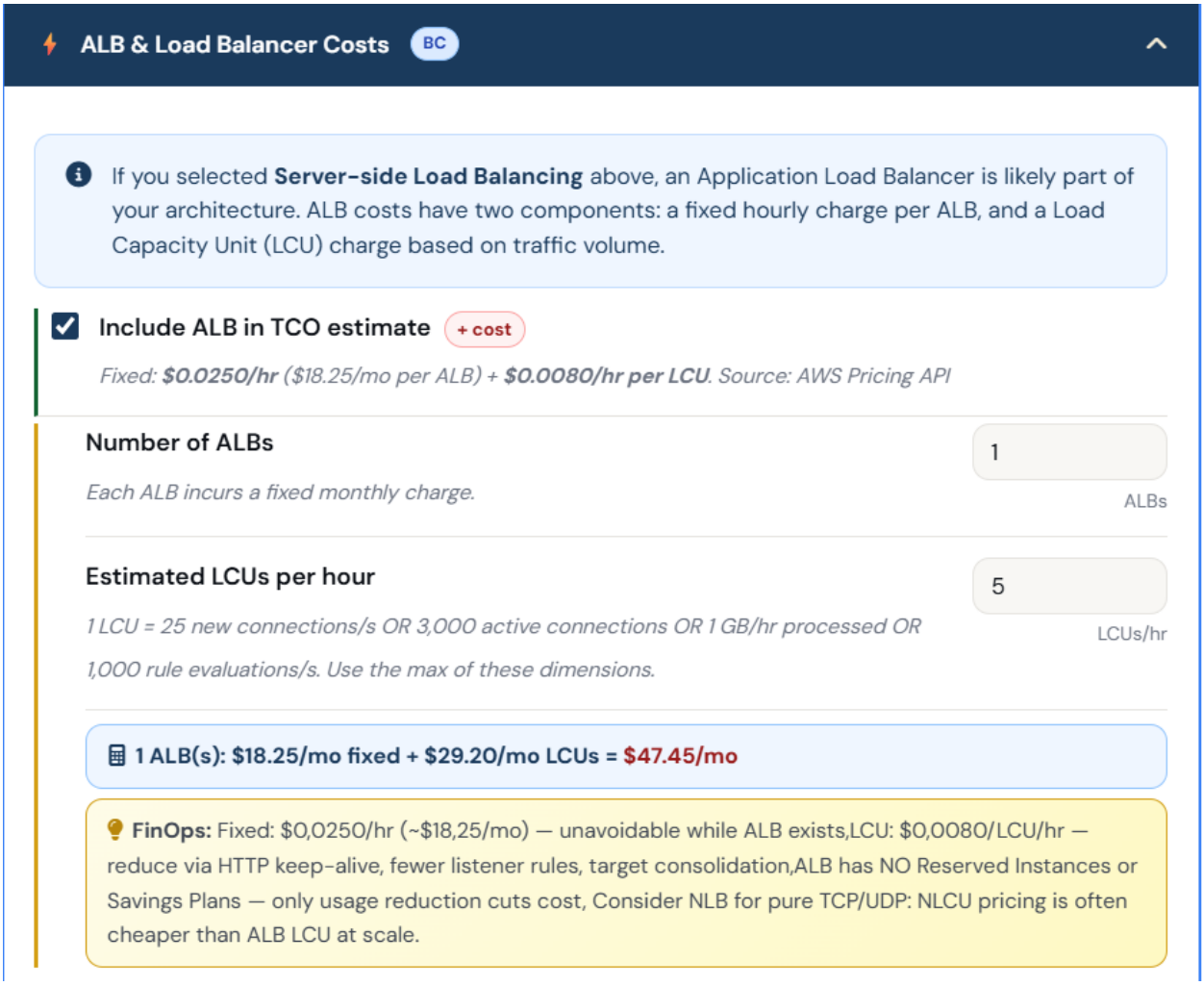


Figure 4.2: End-to-end model -Activity 3: Choose Cloud tactics and patterns mapped to architectural tactics - Server side load balancing mapped to AWS ALB.

Activity 4: Calculate the Base Costs Using Cloud Pricing Calculators - User Story: Base Cost Calculation

Feature: Base Cost Calculation
As a Software Architect
So that I can provide accurate budget forecasts to stakeholders
I want to calculate base costs of my architectural decisions

Scenario Outline: Base cost calculation for different **gRPC-based MACH architecture** configurations
Given I have selected *"<cloud_service>"* to implement *"<tactic>"*
And my base cost provider is *"<cloud_provider>"*
When I trigger the base cost calculation
Then the system should show *"<hourly_cost>"* estimates
And highlight *"<cost_factors>"* as primary cost drivers

Base Cost Calculation Strategy

cloud_provider	cloud_service	tactic	hourly_cost	cost_factors
AWS	Elastic Load Balancer - ALB Layer 7	Server-side Load Balancing	$0.0225	Standard ALB running instances are billed at $0.0225 per hour, combined with a volumetric usage rate of $0.008 per Load Balancer Capacity Unit (LCU) consumed per hour.

Table 4.5: User Story: Base Cost Calculation - Feature Examples

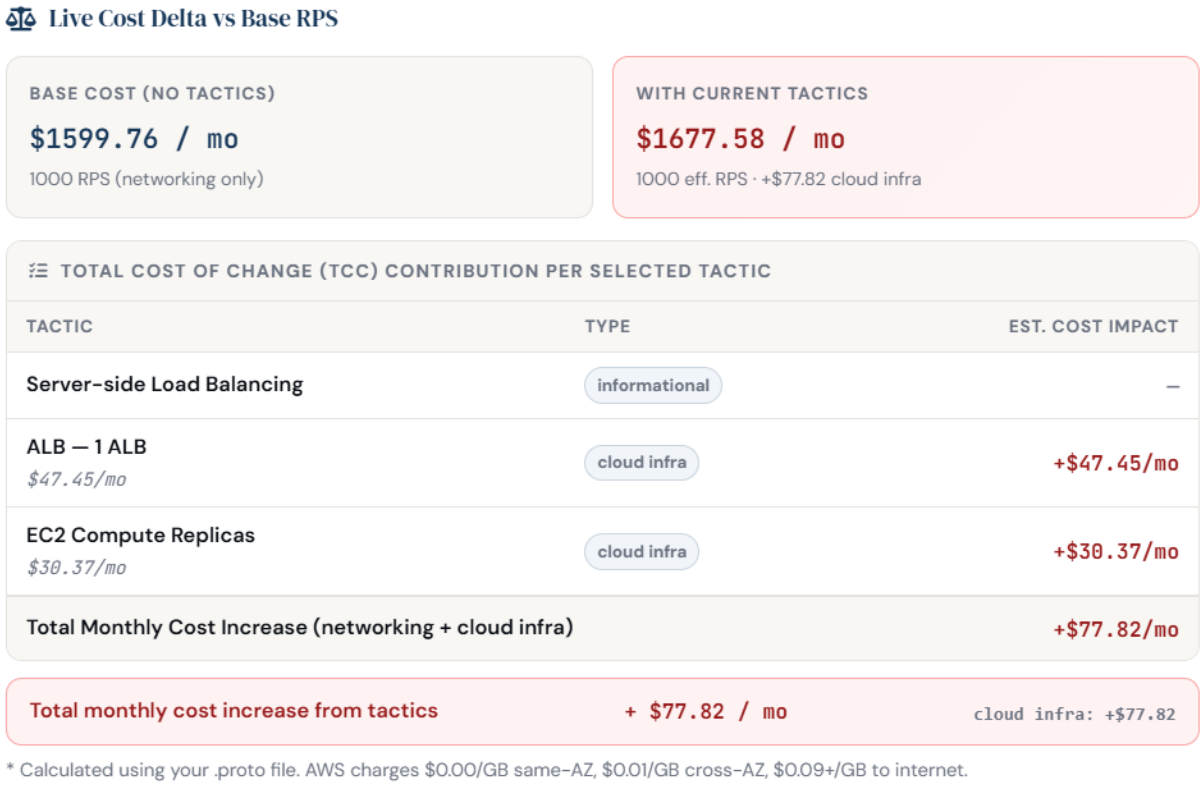


Figure 4.3: End-to-end model - Activity 4: Base costs vs Live Costs.

The previous SBE specification is related to the functionality of verifying a contrast between base costs (considering only networking costs as a result of loading the protocol file) and networking costs plus cloud tactics costs. This can be borne out in phase 3; the tool automatically presents that for each tactic and pattern selection to the Software Architect. As a result, the Software Architect is conscious of the marginal cost of selecting a tactic or pattern. The references used for Hourly Cost Calculations were the following:

Cloud Provider: Amazon Web Services (AWS)

Service/Calculator	Reference	Used for
AWS Pricing Calculator	(Amazon Web Services, 2025a)	Elastic Load Balancer, App Mesh, API Gateway, Certificate Manager
Elastic Load Balancing Pricing	(Amazon Web Services, 2024c)	Elastic Load Balancer

Table 4.6: References used for Hourly Cost Calculations - Cloud Provider: AWS

Activity 5: Define Cost Optimization Strategies (Rate/Utilization) - User Story: Cost Optimization Strategy Definition

Feature: Cost Optimization Strategy Definition
As a Software Architect
So that I can reduce operational expenses while maintaining quality
I want to apply FinOps cost optimization strategies

Scenario Outline: FinOps strategy impact on cloud services implementation
Given a cloud service *"<cloud_service>"* and a *"<cloud_instance>"* implementation with baseline cost *"<base_monthly_cost>"*
When I apply the cost optimization strategy *"<finops_strategy>"*
Then the system should predict a savings percentage of *"<savings_percentage>"* in cost reduction
And indicate the impact of *"<impact>"* on the original tactic's effectiveness.

Cost Optimization Strategy Mapping

cloud_service	cloud_inst	month_cost	finops_stra	savings_perc	impact
AWS EC2	t3.medium	$30.37	Reserved Instances	36%	Reserved Instances (RIs) suit predictable, constant workloads.

Table 4.7: User Story: Cost Optimization Strategy Definition - Feature Examples

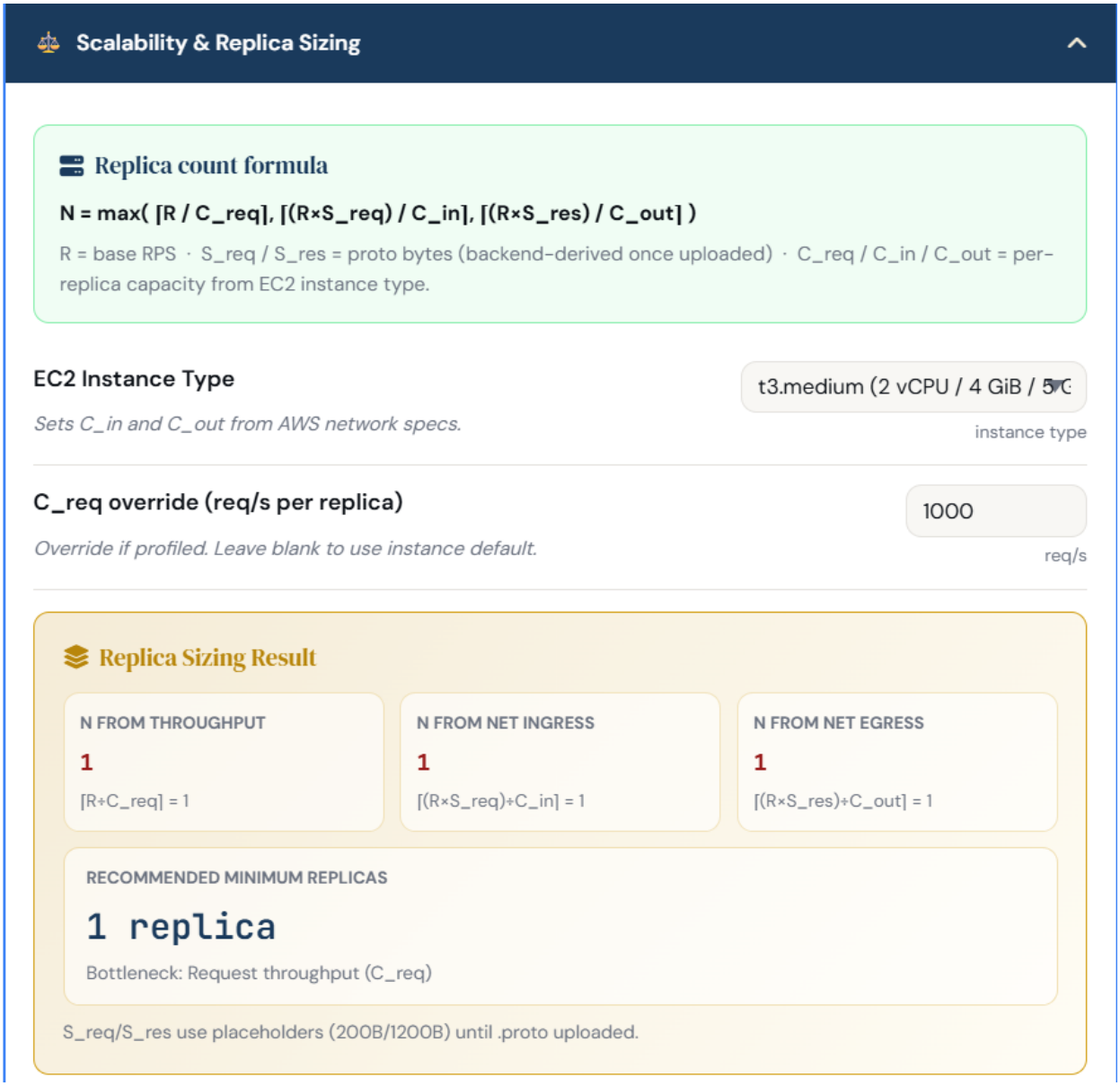


Figure 4.4: Activity 5 - Define cost optimization strategies - EC2 selection.

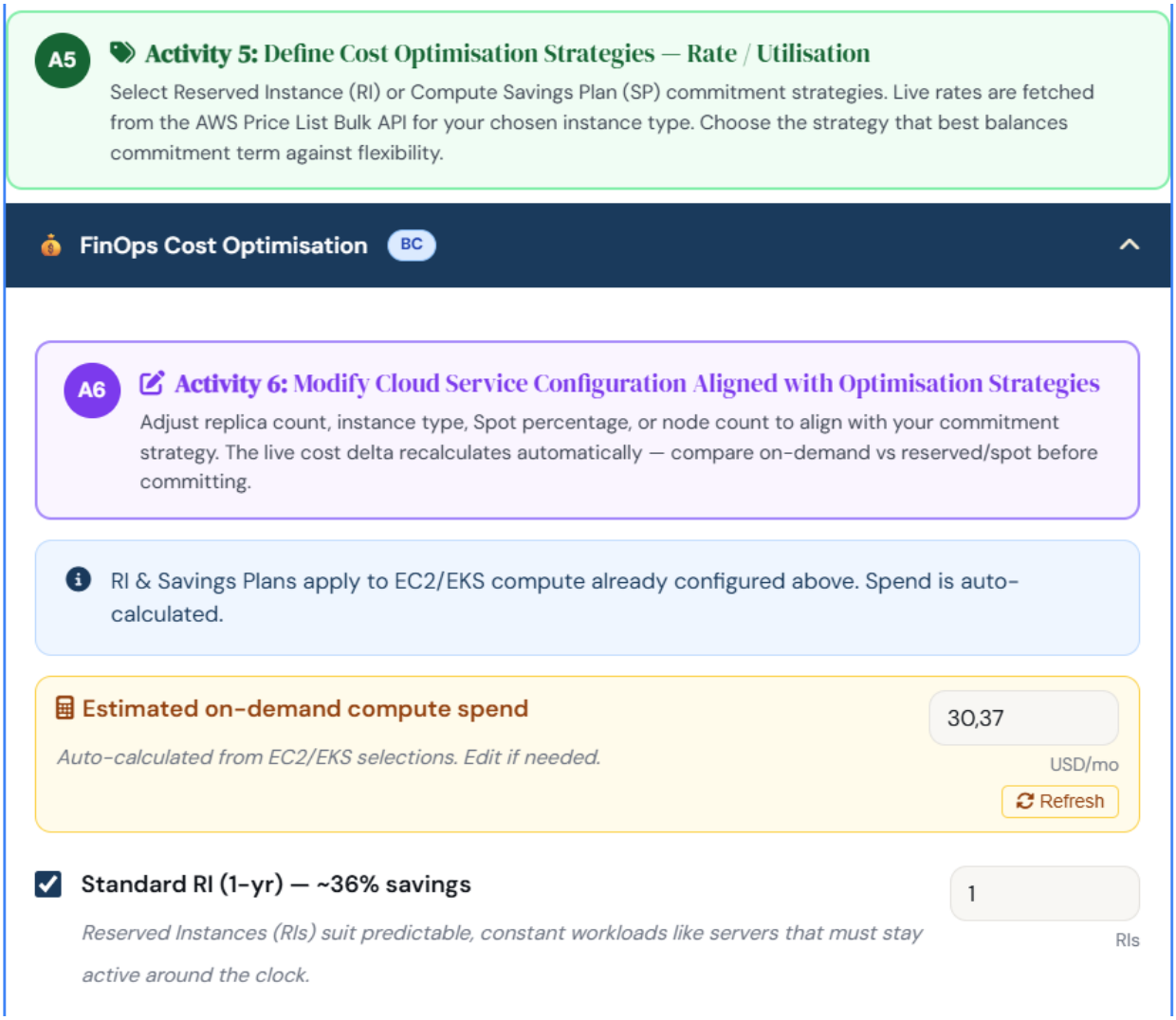


Figure 4.5: Activity 5 - Define cost optimization strategies - Reserved Instances for EC2.

Activity 6: Modify Configuration of Cloud Services Aligned with Cost Optimization Strategies - User Story: Cloud Service Configuration Modification

Feature: Cloud Service Configuration Modification
As a Software Architect
I want to modify cloud service configurations based on optimization strategies
So that I can implement my design with optimal cost-efficiency

Scenario Outline: Optimizing cloud service configurations for cost
Given I have implemented *"<cloud_service>"* and a *"<cloud_instance>"*
When I modify the configuration with *"<finops_strategy>"*
Then the system should calculate *"<monthly_savings>"*

Cloud Service Configuration Tuning

cloud_service	cloud_instance	finops_strategy	monthly_savings
AWS EC2	t3.medium	Reserved Instance 1 year	$10.93

Table 4.8: User Story: Cloud Service Configuration Modification - Feature Examples

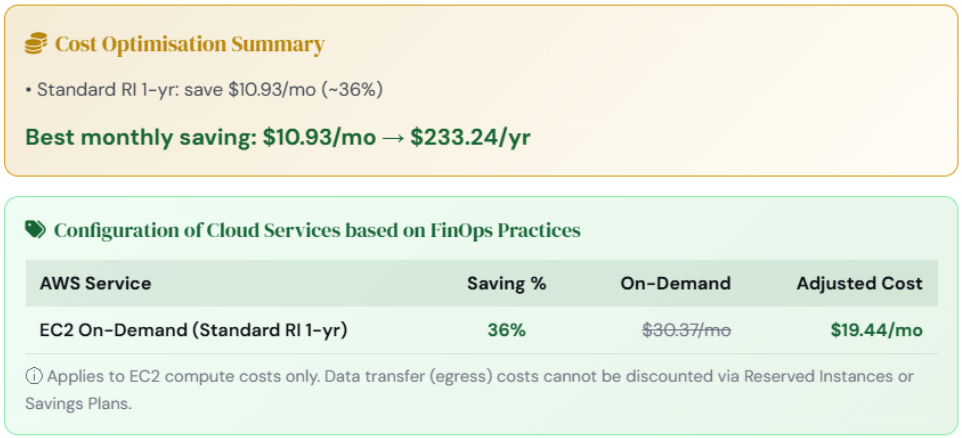


Figure 4.6: Activity6 - Modify configuration of cloud services aligned with cost optimization strategies - Reserved Instaces 1 year for EC2

Activity 7: Calculate the New Cost with the New Configuration - User Story: Optimized Cost Calculation

Feature: Optimized Cost Calculation
As a Software Architect
I want to calculate costs after applying optimization strategies
So that I can quantify financial impact and present cost savings

Scenario Outline: Cost impact for FinOps strategy across cloud service
Given I have applied *"<finops_strategy>"* to services implementing *"<cloud_service>"* and a *"<cloud_instance>"*
When I recalculate total costs across the architecture
Then the system should show *"<new_monthly_cost>"* as the optimized cost
And calculate *"<annual_savings>"*

Optimized Financial Projections

finops_strategy	cloud_service	cloud_instance	new_month_cost	annual_savings
Reserved Instance 1 year	AWS EC2	t3.medium	$19.44	$131.18

Table 4.9: User Story: Optimized Cost Calculation - Feature Examples

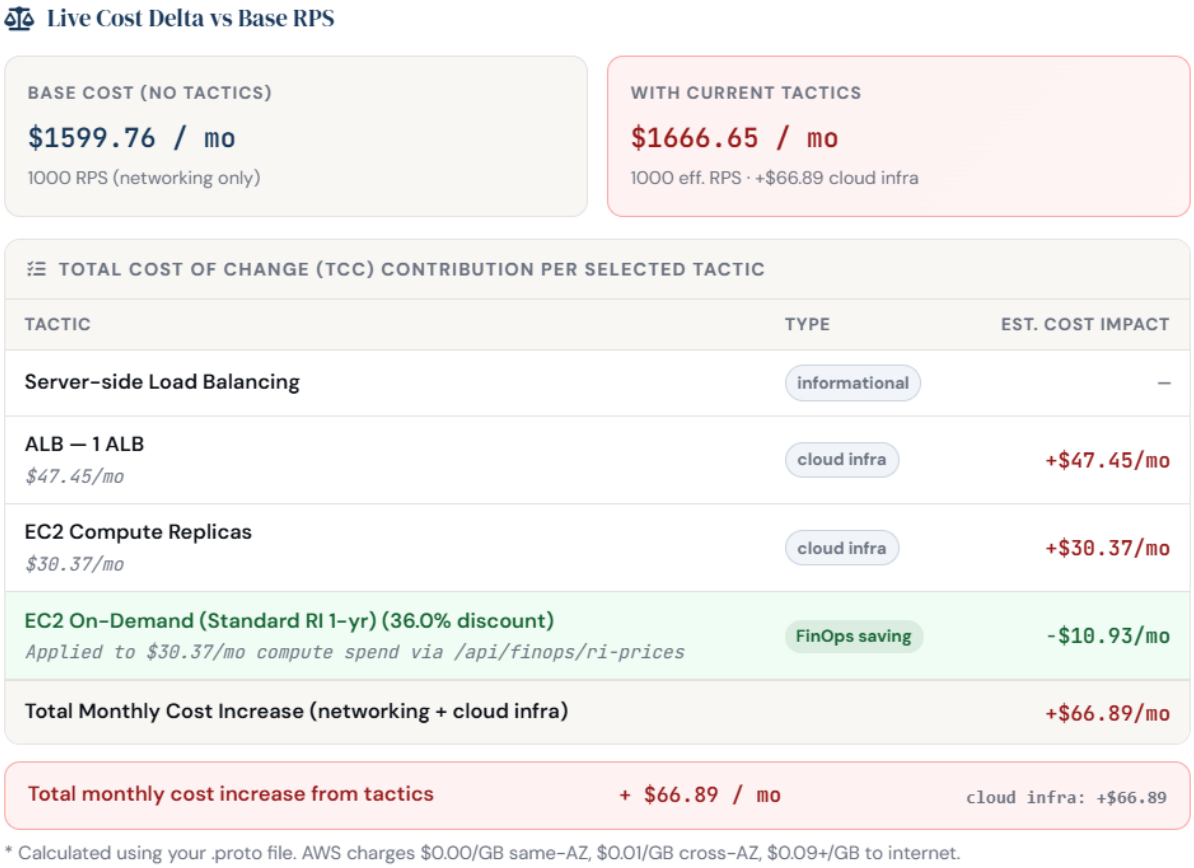


Figure 4.7: Activity 7 - Total cost of change with EC2 and Reserve Instance 1 year.

The rest of the SBE specifications for resiliency and security architectural characteristics are included in the annexes section of this document. Those SBE specifications also contemplate the functionality provided in the different phases. Furthermore, the last two phases are related to the calculation of the whole TCO for the microservice and the FinOps Allocation strategy suggestions that the architect should apply in practice, with an option at the end to add the service cost-efficiency estimation to a portfolio, which, after iterations, has its own view with all the MACH microservices calculated.

Finally, the next section presents the proposed software architecture for the cost-efficiency calculator, in which the SBE living documentation is included.

4.2 Software Architecture for the Cost-efficiency calculator

This final section of the implementation chapter provides a detailed rationale and description of the software architecture behind the cost-efficiency calculator. I followed the architecture design process defined by Cervantes (2016), and I took into account suggestions provided by Clements

(2011) for documenting this section.

The **primary functionality** considered for the architectural design was the one described by the SBE specifications presented in the section above, aligned with the **design purpose** of generating a proof of concept tool for those specifications.

Furthermore, I present the **quality attributes** that acted as **architectural drivers**, along with their **quality attribute scenarios** (Bass et al., 2021).

1. **Extensibility**: which determines *"how important it is to plug new pieces of functionality in."* (Richards, Lange, & Ford, 2021), and as AWS suggests, *"make sure the system is extensible enough"* (Kumar & Singh, 2025). The calculator was designed to plug multiple cloud providers into the calculator in the future. This is aligned with the domain concern: Mergers and Acquisitions (Richards et al., 2021), which is crucial for the evolution of the cost-efficiency calculator. **Extensibility Scenario**:
   - **Source**: a software architect using the cost-efficiency calculator.
   - **Stimulus**: the software architect calculates the cost-efficiency of its gRPC-based MACH microservice, considering one cloud provider.
   - **Artifact**: cost-efficiency calculator. **Environment**: normal operation mode.
   - **Response**: the calculator calculates the cost-efficiency based on the parametrized cloud provider (AWS for the scope of this proof of concept tool).

   For this quality scenario, I decided to use **ports and adapters** as an **architectural pattern** to extend the functionality to makes calls to another pricing calculators of other cloud providers like Microsoft Azure and Google Cloud platform in the future, following San Miguel (2024) suggestions.
2. **Configurability**: which is defined as *"the ability for the end users to easily change aspects of the software's configuration (through usable interfaces)"* (Richards et al., 2021). This enables precise cost calculation, predictive forecasting, architectural trade-off parametrization, unit economics calculation, and MACH portfolio management. **Configurability Scenario**:
   - **Source**: a software architect using the cost-efficiency calculator.
   - **Stimulus**: the software architect selects an architectural tactic or pattern, or a cloud service or a FinOps strategy to apply for its gRPC-based MACH microservice.
   - **Artifact**: cost-efficiency calculator. **Environment**: normal operation mode.
   - **Response**: the calculator should provide, for each architectural pattern and tactic, and for each cloud service and FinOps optimization strategy, a way of specifying their different parameters to reach a more accurate cost-efficiency estimation.
   - **Response Measure**: the user interface should display different options for parameterizing those criteria and, based on that, calculate the cost-efficiency.

   The calculator supports **different parameters in the UI to be specified by the software architect** for FinOps strategies, cloud services, and architectural patterns, and tactics like the retry failure percentage (5% by default).

3. **Feasibility**: which means *"taking into account timeframes, budgets, and developer skills when making architectural choices; tight timeframes and budgets make this a driving architectural characteristic"* (Richards, 2024). This is aligned with the domain concern: Time and budget (Richards et al., 2021), which were tight constraints for this system. **Feasibility Scenario**:
   - **Source**: a software architect using the cost-efficiency calculator.
   - **Stimulus**: the software architect requests the calculation of the cost-efficiency of the gRPC-based microservice of its MACH architecture.
   - **Artifact**: cost-efficiency calculator.
   - **Environment**: normal operation mode.
   - **Response**: The cost-efficiency calculation should consider only Amazon Web Services as cloud provider and the system should deliver both TCO and unit economics.

The calculator uses the **AWS PricingClient in each of the AWS adapters** to requests for the AWS Pricing of the different services like Amazon EC2, EKS, or FinOps strategies like reserved instances (Amazon Web Services, 2026j; MvnRepository, 2026).

Finally, another constraint aligned with **feasibility** is the support of only the ProtocolBuffer files defined by each of the four business use cases for the gRPC communication patterns that were defined in the annex section. No other protocol buffer files with a different structures were supported and need to be included as future work.

4.2.1 Architecture Styles and Patterns

As Mark Richards and Neal Ford explain, *"we define an architectural style as the overarching structure of how the user interface and backend source code are organized [...] and how that source code interacts with a datastore."* (Richards et al., 2021). For the cost-efficiency calculator, I chose the **layered architecture**, which is a *"monolithic (single deployment unit of all code)"* (Richards et al., 2021), that *"is good choice for small, simple applications or web sites. It is also a good architecture choice, particularly as a starting point, for situations with very tight budget and time constraints."* (Richards et al., 2021). My criteria was aligned with **feasibility** as *"Overall cost and simplicity are the primary strengths of the layered architecture style"* (Richards et al., 2021).

The following are the implemented **layers** (as **architectural elements**). In the annex section there are more details for the layers structure in the GitHub repository.

1. The **Application Layer** contains the calculator ports, the effective requests per second calculators, and the protocol buffer file compilators and services, among others like mappers and tactics populators.

2. The **Domain Layer** contains the data transfer objects used for requests and responses against the controllers defined in the infrastructure layer as well as responses obtained by calling methods in other services. It includes the end-to-end model classes and the repositories for all architectural decisions (tactics, patterns, cloud services and FinOps strategies).

3. The **Infrastructure Layer**: contains both the AWS adapters and the controllers including the Model-view-controller ones.

4. The **Shared Layer**: contains tools like the protocol buffer file parser, JSON response loggers, and another protocol buffer utilities.

On the other hand, the **architectural patterns** implemented in this layered monolith are as follows:

1. **Ports & Adapters (a.k.a Hexagonal Architecture)** because it allows us to link applications together without any user involvement (Cockburn, 2005). Through ports and adapters, it provides the extensibility that the calculator needs to use other cloud vendors, architectural characteristics, and tactics and patterns in the future.
2. **Model-view-controller** for allowing the software architect to use a UI view to perform all its activities, and delegating to the controllers the calculations. I selected Thymeleaf (2026) as a view technology rather than more modern frameworks like Angular, because it only requires to be deployed in the same container as the backend service in the MVC pattern approach defined by Spring (Borowiec, 2026; Broadcom, 2026). That avoided me to have two containers to be deployed, one for the frontend and another one for the backend, and it perfectly aligned with the **feasibility** architectural driver.

4.2.2 Software Architecture C4 diagrams

For diagramming the software architecture **views**, I chose the **C4 model**. Brown (2025) presents it as an approach for visualizing software architectures in different levels (hierarchy of abstractions) depending on the target audience. This allowed me to avoid clutter and confusion (Read, 2023).

Moreover, Brown (2026) is also the creator of **Structurizr**, which is a tool that uses diagrams as code, allowing me to create multiple software architecture diagrams using DLS language. This approach was used to diagram the different architectural views in this section. Therefore, the following are the different **architectural views**.

In the Context diagram, it is shown that the Software Architect is the person who evaluates the cost-efficiency of a gRPC-based MACH architecture by analyzing trade-offs across resiliency, reliability, and security. Furthermore, the Cost-efficiency calculator uses the Cloud Provider APIs to obtain the services' pricing and FinOps strategies and commitment discounts.

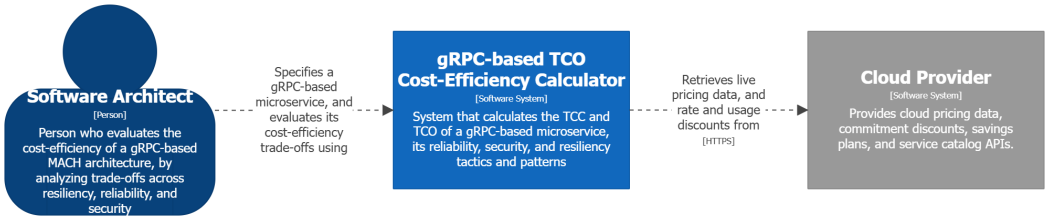


Figure 4.8: Cost-efficiency calculator - C4 Model: Context Diagram. Generated with Structurizr DSL.

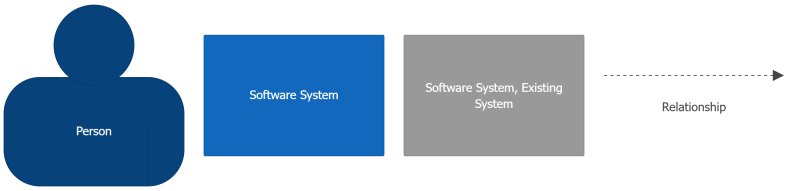


Figure 4.9: C4 model - System Context Notation specification.

Next, in the following containers diagram, the cost-calculator system is shown to be deployed as a monolith in a container. To develop this service, the spring boot project of the Java Spring ecosystem is specified.

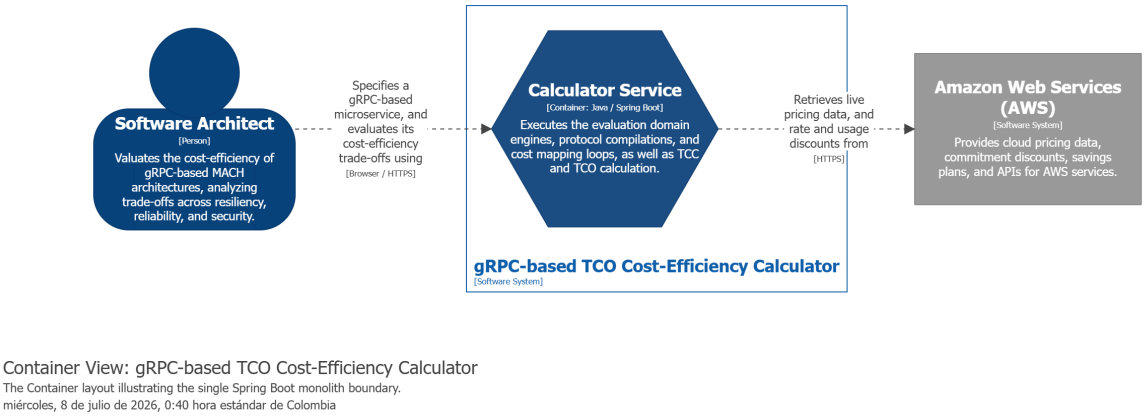


Figure 4.10: Cost-efficiency calculator - C4 Model: Containers Diagram. Generated with Structurizr.

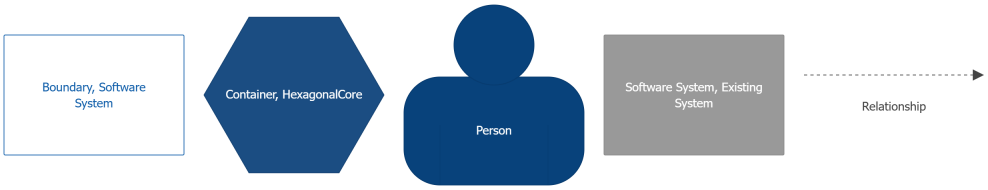
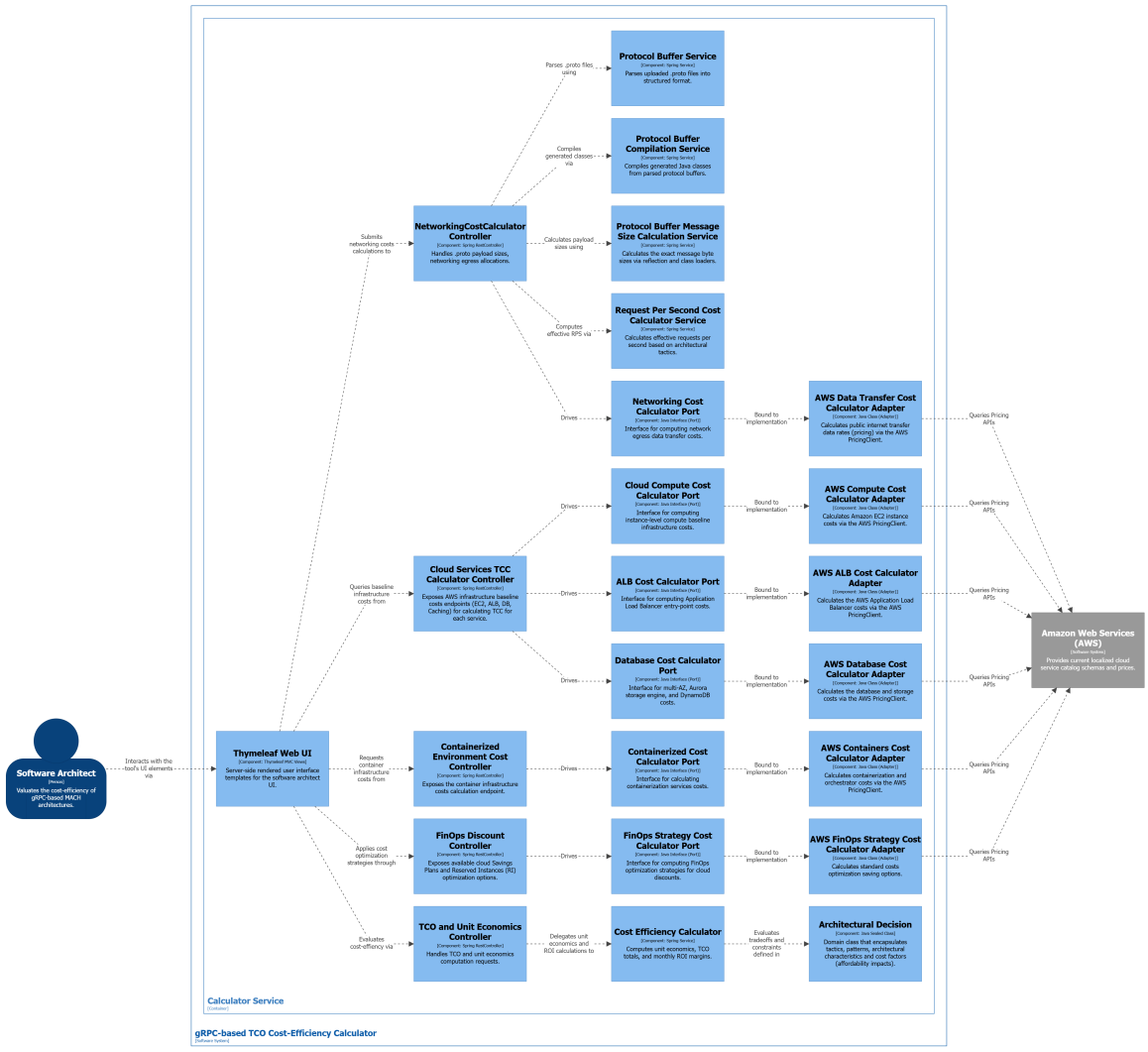


Figure 4.11: C4 model - Containers Notation specification.



Component View: gRPC-based TCO Cost-Efficiency Calculator - Calculator Service
Pure Hexagonal View laid out sequentially from Left to Right with direct lines.
lunes, 20 de julio de 2026, 18:51 hora estándar de Colombia

Figure 4.12: Cost-efficiency calculator - C4 Model: Components Diagram. Generated with Structurizr DSL.

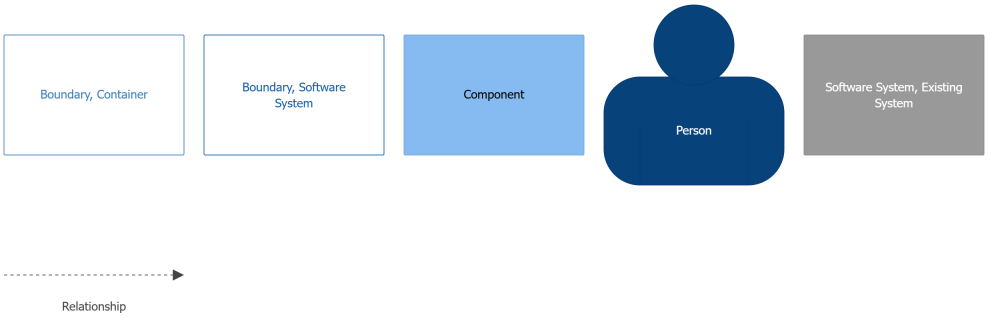


Figure 4.13: C4 model - Component Notation specification.

Finally, in this component diagram, I present the controllers to receive the software architect requests from the UI, the ports, and adapters used to make AWS API requests to the AWS cloud provider, and to perform the calculations of Networking, TCC and TCO costs, as well as the service to conduct the unit economics calculations. Furthermore, I defined the system components and their inbound/outbound relationships as follows:

1. **Thymeleaf Web UI (View of the Mode-View-Controller):** Provides server-side rendered user interface templates for the software architect UI. It serves as the primary external entry point, routing user interactions directly to the inbound API controller adapters.

2. **Inbound REST Controllers (Infrastructure Layer):** These controllers serve as primary adapters, capturing external HTTP commands and driving the internal domain ports or services:

   - **NetworkingCostCalculatorController:** Handles `.proto` payload sizes, networking egress costs by driving the `NetworkingCostCalculatorPort`.
   - **Cloud Services TCC Controller:** Exposes endpoints for AWS infrastructure baseline costs (including Amazon compute, ALB, and storage), driving the compute, load balancer, and database ports.
   - **Containerized Environment Cost Controller:** Exposes the container infrastructure costs calculation endpoint, driving the `ContainerizedCostCalculatorPort`.
   - **FinOps Discount Controller:** Exposes available cloud Savings Plans and Reserved Instances (RI) optimization options, driving the `FinOpsStrategyCostCalculatorPort`.
   - **TCO and Unit Economics Controller:** Handles TCO and unit economics computation requests and evaluates business viability constraints by delegating the workflow to the internal application service layer: the **CostEfficiencyCalculator** service and all the architectural decisions chosen by the software architect through the UI.

3. **Application Core Ports and Services (Application Layer):** This pure domain layer orchestrates abstract capabilities and cost evaluation contracts:

- **Core Calculator Ports:** Define abstract Java interfaces—including *CloudComputeCostCalculatorPort*, *ALBCostCalculatorPort*, *DatabaseCostCalculatorPort*, *FinOpsStrategyCostCalculatorPort*, *ContainerizedCostCalculatorPort*, and *NetworkingCostCalculatorPort*—which isolate business computations from infrastructure-specific libraries.
- **Cost Efficiency Calculator:** A core Spring service driven by the TCO and Unit Economics Controller. It computes unit economics, total TCO costs, and monthly ROI margins.

4. **Domain classes and Outbound Adapters (Domain Layer and Infrastructure Layer):** Implement concrete business rules and map core boundaries to external providers:

   - **Architectural Decision:** A domain class implemented as a Java Sealed Class, encapsulating tactics, patterns, architectural characteristics, cloud services and finops strategies, each of which having their cost factors or affordability impacts.
   - **Infrastructure Cost Adapters:** Concrete classes—including the AWS Compute, ALB, Database, FinOps Strategy, Containers, and Data Transfer adapters—that bind tightly to core interfaces and interact directly with the external Amazon Web Services (AWS) Pricing APIs to pull cloud service catalog schemas and real-time prices via the **AWS PricingClient**.

4.3 Chapter Summary

In this chapter, the cost-efficiency calculator implementation was described by identifying the SBE specifications, the software architect use cases with their related Graphical User Interface demonstrations, the software architectural characteristics, the architectural style and patterns, as well as the C4 model.

In the next chapter, the evaluation of the End-to-end model and the cost-efficiency calculator tool for software architects is presented.

Chapter 5

Evaluation

"I shall not fail, finally, to warn that I do not claim to believe that this work has turned out either complete or perfect, but in any case I have the hope that it may become a not useless guide for the scholar who wishes to deepen further the studies of analysis."

Ernesto Pascal, Esercizi E Note Critiche Di Calcolo Infinitesimale, Pavia, March, 1895.
Translated from (Pascal, 1895)

Evaluating the end-to-end model was crucial for determining the completeness, coherence, and perceived usefulness of the End-to-end model and the cost-efficiency calculator tool for software architects. To do so, Software Engineers, Architects, Team Leads, DevOps professionals, and FinOps Foundation representatives were contacted to conduct evaluation sessions following the same criteria. This section presents the results obtained in two phases of evaluation: **KR1 - To conduct and document a qualitative evaluation of the proposed cost estimation model**, focusing on perceived usefulness, intention to use, and usability based on expert feedback from professionals in cloud architecture and software design; and **KR2 - To conduct and document a qualitative evaluation of the proposed practical tool**, focusing on perceived usefulness, intention to use, and usability based on expert feedback from professionals in cloud architecture and software design.

The evaluation of the **end-to-end model** was based on the following **criteria**:

Evaluation Criteria Definitions

Criteria	Definition
Completeness	Addresses the defined project scope and objectives; ensures all relevant aspects of FinOps, MACH, and gRPC are covered without leaving critical gaps.
Coherence	Alignment with principles, patterns, and practices. Evaluates logical consistency, adherence to best practices, and whether components work together cohesively.
Perceived Usefulness	Value for software architects and stakeholders in solving real-world problems. Evaluates practicality, applicability, and benefits in designing and calculating cost-efficiency.
Consistency	Adherence to standards and data integrity. Ensures predictable cost management (FinOps), modularity/interoperability (MACH), and reliable communication protocols (gRPC).

Table 5.1: Evaluation criteria definitions.

In order to do so, the following virtual meeting (40 minutes) structure was defined for each evaluation session with stakeholders: Project's Problem & Scope (5 minutes), Proposed model (10 minutes), Q&A Feedback (15 minutes), Validation of the information received (5 minutes). Conclusions and next steps (5 minutes).

To mitigate the inherent risk of misinterpretation that can arise during evaluations conducted through meetings. These strategies include paraphrasing key points during discussions to ensure mutual understanding, recording meetings via Zoom, Notta.ai, for accurate reference, utilizing AI-powered transcription services to create a detailed written record, and requesting a concise evaluation summary to be delivered via direct message, thus providing multiple layers of verification and documentation to enhance the reliability of the evaluation outcomes. As well as direct requests in message channels from which screenshots are taken and then attached as images for each development phase.

The evaluation survey proposed to evaluate the conceptual model was the following.

Conceptual Model Evaluation Survey

Criteria	Description	Scale (1-5)
Completeness		
Domain Coverage	How comprehensively does the class diagram represent the domain of microservices evaluation, gRPC patterns, and FinOps?	1: Not comprehensive at all, 5: Comprehensive
Scope Alignment	How well does the conceptual model align with the project's scope on cost efficiency, security, reliability, and resiliency?	1: Not aligned at all, 5: Well aligned
MACH Representation	How effectively does the model represent MACH principles to support evaluation?	1: Not effective at all, 5: Very effective
Coherence		
FinOps Conceptualization	How effectively does the conceptual model incorporate FinOps principles and practices into its structure?	1: Not effective at all, 5: Very effective
gRPC Patterns	How clearly and coherently are gRPC patterns (Unary, Streaming, Bidirectional) represented in the class diagram?	1: Not clear/coherent at all, 5: Very clear/coherent
gRPC Best Practices	How well does the conceptual model adhere to best practices like health probes and TLS handshakes?	1: Does not adhere at all, 5: Adheres well

Table 5.2: Conceptual model evaluation survey.

By that time, this was the end-to-end model class diagram evaluated version.

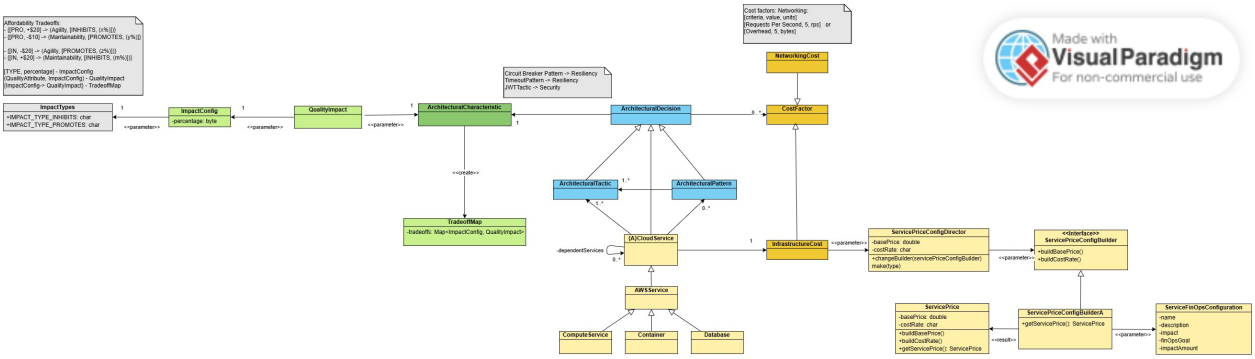


Figure 5.1: End-to-end-Model-Class-Diagram.

5.0.1 Evaluation results by Stakeholder's role: Senior Software Engineering Manager

I contacted him via Slack to have a conference with him. He has experience in working in software architectures supporting e-commerce operations. The following table shows his evaluation results and observations.

Survey Results: Architecture Evaluation

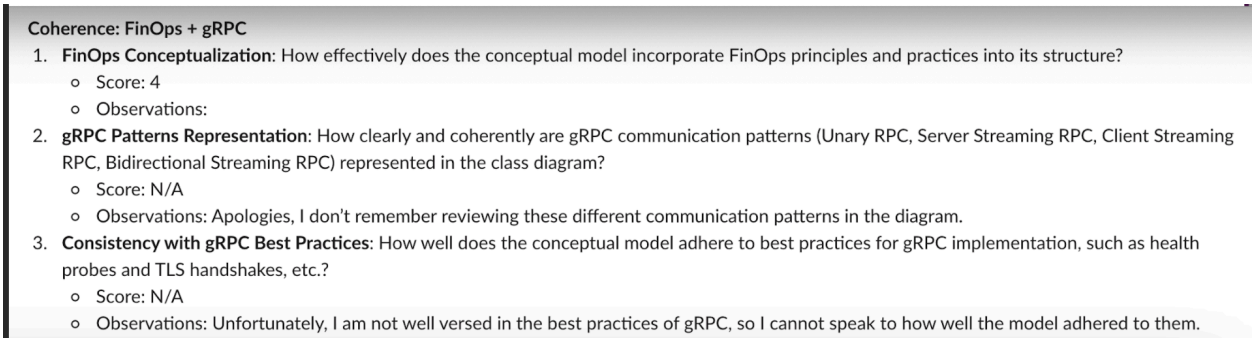
Criteria	Observations	Scale
Completeness		
Domain Coverage	*"I would have liked to drive into the concept a bit more at a summary view so that the architecture diagrams were understandable at an earlier state of the presentation. We ultimately got there and the diagrams were well representative of the proposal".*	4
Scope Alignment	*"The model was really impressive and left us talking about why this functionality didn't already exist. It would be incredible to see something like this available".*	5
MACH Architecture	No additional observations.	4
Coherence		
FinOps Conceptualization	No additional observations.	4
gRPC Patterns	*"Apologies, I don't remember reviewing these different communication patterns in the diagram".*	N/A
gRPC Best Practices	*"Unfortunately, I am not well versed in the best practices of gRPC, so I cannot speak to how well the model adheres to them".*	N/A
Perceived Usefulness		
"The model was really impressive and left us talking about why this functionality didn't already exist. It would be incredible to see something like this available".		

Table 5.3: Survey results: architecture evaluation.

Completeness: Problem Definition + Project's Scope 1 (Not clear or coherent at all) to 5 (Very clear and coherent)

1. **Domain Coverage**: How comprehensively does the class diagram represent the domain of microservices architecture evaluation, including gRPC communication patterns and FinOps practices?
   - Score: 4
   - Observations: I would have liked to dive into the concept a bit more at a summary view so that the architecture diagrams were understandable at an earlier state of the presentation. We ultimately got there and the diagrams were well representative of the proposal.
2. **Scope Alignment**: How well does the conceptual model align with the project's scope, focusing on cost efficiency, security, reliability, and resiliency in microservices architectures?
   - Score: 5
   - Observations: The model was really impressive and left us talking about why this functionality didn't already exist. It would be incredible to see something like this available.
3. **MACH Architecture Representation**: How effectively does the conceptual model represent the MACH architecture principles to support the evaluation of microservices architectures?
   - Score: 4

Figure 5.2: Senior Software Engineering Manager's evaluation results part I.



Coherence: FinOps + gRPC

1. **FinOps Conceptualization**: How effectively does the conceptual model incorporate FinOps principles and practices into its structure?
   - Score: 4
   - Observations:
2. **gRPC Patterns Representation**: How clearly and coherently are gRPC communication patterns (Unary RPC, Server Streaming RPC, Client Streaming RPC, Bidirectional Streaming RPC) represented in the class diagram?
   - Score: N/A
   - Observations: Apologies, I don't remember reviewing these different communication patterns in the diagram.
3. **Consistency with gRPC Best Practices**: How well does the conceptual model adhere to best practices for gRPC implementation, such as health probes and TLS handshakes, etc.?
   - Score: N/A
   - Observations: Unfortunately, I am not well versed in the best practices of gRPC, so I cannot speak to how well the model adhered to them.

Figure 5.3: Senior Software Engineering Manager's evaluation results part II.

5.0.2 Evaluation results by Stakeholder's role: Tech Lead Manager

I contacted him via Slack to have a conference with him. He has experience in working in software architectures supporting e-commerce operations. The following table shows his evaluation results and observations.

Tech Lead Manager Evaluation Results

Criteria	Observations	Scale
Completeness		
Domain Coverage	No additional observations.	5
Scope Alignment	No additional observations.	5
MACH Architecture	No additional observations.	5
Coherence		
FinOps Conceptualization	No additional observations.	5
gRPC Patterns	No additional observations.	3
gRPC Best Practices	No additional observations.	4
Perceived Usefulness		
"Very interesting presentation, it would be definitely something that could help organizations when setting up new infra. To take a different approach and have another outlook on what would resources would be best to use" .		

Table 5.4: Tech Lead Manager evaluation results.

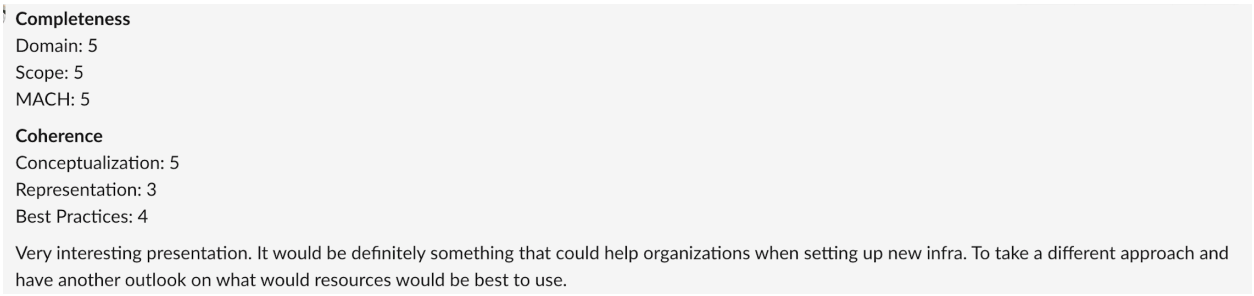


Figure 5.4: Tech Lead Manager's evaluation results

5.0.3 Evaluation results by Stakeholder's role: Senior Software Development Engineer

I contacted him via Slack to have a conference with him. He has experience in working in software architectures supporting e-commerce operations. The following table shows his evaluation results and observations.

Feedback: Senior Software Development Engineer
"I can see William implemented a very detailed architecture graph that would ensure the resiliency and robustness of the health of system for financial cloud tech. The use of various resilience and deployment techniques could help ensure system high availabilities, it's overall a comprehensive design".

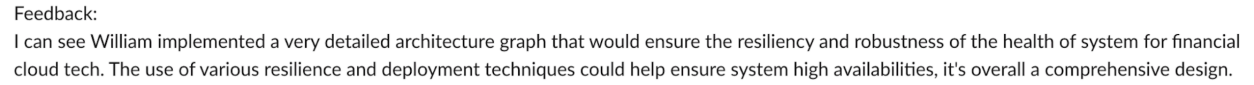


Figure 5.5: Senior Software Development Engineer's evaluation results.

5.0.4 Evaluation results by Stakeholder's role: DevSecOps Lead

I contacted him via LinkedIn and via email to have a meeting and a point of view from his DevOps experience in the software industry. The following table shows his evaluation results and observations.

DevSecOps Lead survey Results: Model Evaluation

Criteria	Observations	Scale
Completeness		
Domain Coverage	Consider the cost of interactions between services and variations due to interconnection (IPs), and region variations (same region or multi-region), for Future Work.	5
Scope Alignment	Evidence of a formula for cost calculation is presented.	5
MACH Architecture	Highly effective, yet the persistence tactics and patterns are not clearly demonstrated.	5
Coherence		
FinOps Conceptualization	To illustrate the unit economics of FinOps, examining how infrastructure changes affect business revenue or ROI, and to establish these unit economics in terms of costs per request/second or per request package	4
gRPC Patterns	No additional observations.	5
gRPC Best Practices	A telemetry use case in which traces, logs and metrics are analyzed in Prometheus could be a good one.	5

Table 5.5: DevSecOps Lead survey results: model evaluation.

On the other hand, to conduct and document a qualitative evaluation of the proposed practical tool, the following questions were proposed to evaluate completeness, coherence, perceived usefulness, and consistency. By that time, the end-to-end model class diagram evaluated version was the one last one presented already in the System Design section of the Project Development chapter in this document.

Survey Template: Cost-efficiency calculator Tool Evaluation

Criteria	Description	Scale
Completeness: Problem Definition + Project's Scope		
Domain Coverage	How comprehensively does the tool represent microservices architecture, gRPC, and FinOps?	1 (Not comprehensive) to 5 (Comprehensive)
Scope Alignment	How well does the interface align with scope on cost, security, reliability, and resiliency?	1 (Not aligned) to 5 (Well aligned)
MACH Representation	How effectively does the tool represent MACH principles?	1 (Not effective) to 5 (Very effective)
Coherence: FinOps + MACH + gRPC		
FinOps Conceptualization	How effectively does the tool integrate FinOps principles into its 5-phase structure?	1 (Not effective) to 5 (Very effective)
gRPC Patterns	How clearly are gRPC patterns (Unary, Streaming) represented in Phase 2?	1 (Not clear) to 5 (Very clear)
gRPC Best Practices	How well does the tool reflect best practices like TLS?	1 (Does not adhere) to 5 (Adheres well)
Perceived Usefulness		
Practical Relevance	How useful is the 5-phase workflow for real-world gRPC cost scenarios?	1 (Not useful) to 5 (Very useful)
Ease of Understanding	How easy is the tool to understand/apply for FinOps engineers?	1 (Difficult) to 5 (Easy)
Decision Support	How effectively does the tool provide support for tradeoffs?	1 (Not effective) to 5 (Very effective)
Consistency		
Consistency	How consistent is the phase-based representation with underlying models?	1 (Not consistent) to 5 (Very consistent)
Standards Alignment	How well does the tool align with FinOps, microservices, and gRPC standards?	1 (Does not align) to 5 (Aligns well)
Scalability	How well does the workflow handle evolving requirements?	1 (Not accommodate) to 5 (Accommodates well)
Tradeoff Framework	How effectively does Phase 3 provide a FinOps framework for tradeoffs?	1 (Not effective) to 5 (Very effective)
Extensibility and Adaptability		
Connectivity	What potential does the tool have for connectivity to TCO, ITSM, or ITIL?	1 (Limited potential) to 5 (High potential)

Table 5.6: Survey template: cost-efficiency calculator tool evaluation.

5.0.5 Evaluation results by Stakeholder's role: FinOps Professional

He is one of the few FinOps Professionals available in LATAM. I contacted him via the FinOps Foundation slack channel and via LinkedIn. The following table shows his evaluation results and observations.

FinOps Professional Evaluation Summary

Criteria	Description	Score
Completeness: Problem Definition + Project's Scope		
Domain Coverage	Comprehensive representation of microservices, gRPC, and FinOps	4
Scope Alignment	Alignment with cost efficiency, security, reliability, and resiliency	4
MACH Representation	Representation of MACH principles in evaluation phases	4
Coherence: FinOps + MACH + gRPC		
FinOps Conceptualization	Integration of FinOps principles (cost allocation) into 5-phase structure	4
gRPC Patterns	Clarity of gRPC patterns (Unary, Streaming) representation	4
gRPC Best Practices	Reflection of best practices like TLS	4
Perceived Usefulness		
General Feedback	The presentation looks very good, is very solid, and is technical.	N/A
Extensibility and Adaptability		
Connectivity	Potential for connectivity (TCO, ITSM, ITIL, multi-cloud)	4

Table 5.7: FinOps Professional evaluation summary.

5.0.6 Evaluation results by Stakeholder's role: FinOps Foundation Representative

He is a FinOps Foundation representative for LATAM. I contacted him via the FinOps Foundation slack channel and via LinkedIn. The following table shows his evaluation results and observations.

Hola, William.

Aquí están mis comentarios de tu herramienta. Avísame si puedo ayudarte con algo más.

Completeness: Problem Definition + Project's Scope
Domain Coverage: 5.
Scope Alignment: 4. As far as I saw, It's pretty aligned with the scope, but I cannot comment on the security aspect.
MACH Architecture Representation: 5.

Coherence: FinOps + MACH Architectures + gRPC
FinOps Conceptualization: 5. It seems perfectly aligned with this topic.
gRPC Patterns Representation: 5.
Consistency with gRPC Best Practices: 5.

Perceived Usefulness
Practical Relevance: 3. I still don't see a practical usability. I dunno if it's a lack of knowledge on my end, but I still think someone can perform the same activity with a simple automation process. This could impact the growth and usage of this tool.
Ease of Understanding 3. I think that the UI need to be improved. Maybe a more friendly UI or a more simple one. Even adding some notes that could guide the user.
Decision Support: 4. Not sure about the security part again, but it seems to properly provide cost, reliability and resilience.

Consistency
Consistency Across Models: 5. Not sure about this question.
Alignment with Industry Standards: 4. I think that it's alligned with FinOps and microservices somehow.
Scalability and Flexibility: 5.

Consistency
Tradeoff Analysis Framework: 4. I can see some FinOps Capabilities, but I don't think it reaches the entire framework.
Data Persistence Strategy: 5.
Extensibility and Adaptability: 5.
•••
–
Grato / Best Regards,

Figure 5.6: FinOps Foundation Representative evaluation proof.

FinOps Foundation Representative Evaluation Summary

Criteria	Details	Score
Completeness: Problem Definition + Project's Scope		
Domain Coverage	How comprehensively does the calculator tool workflow represent the domain of microservices architecture evaluation, including gRPC patterns and FinOps practices?	5
Scope Alignment	As far as I saw, It's pretty aligned with the scope, but I cannot comment on the security aspect.	4
MACH Architecture Representation	How effectively does the tool represent MACH principles across its phases to support microservices evaluation?	5
Coherence: FinOps + MACH Architectures + gRPC		
FinOps Conceptualization	It seems perfectly aligned with this topic.	5
gRPC Patterns Representation	How clearly are gRPC patterns (Unary, Server/Client/Bi-Directional Streaming) represented in Phase 2 of the tool?	5
Consistency with gRPC Best Practices	How well does the tool (e.g., Phases 3-4) reflect gRPC best practices like TLS?	5
Perceived Usefulness		
Practical Relevance	I still don't see a practical usability. I dunno if it's a lack of knowledge on my end, but I still think someone can perform the same activity with a simple automation process. This could impact the growth and usage of this tool.	3
Ease of Understanding	I think that the UI need to be improved. Maybe a more friendly UI or a more simple one. Even adding some notes that could guide the user.	3
Decision Support	Not sure about the security part again, but it seems to properly provide cost, reliability and resilience.	4
Consistency		
Consistency Across Models	Not sure about this question.	5
Alignment with Industry Standards	I think that it's alligned with FinOps and microservices somehow.	4
Scalability and Flexibility	How well does the tool's workflow (e.g., Phase 5 portfolio) handle evolving FinOps/microservices requirements?	5
Consistency (Section 2)		
Tradeoff Analysis Framework	I can see some FinOps Capabilities, but I don't think it reaches the entire framework.	4
Extensibility and Adaptability	What potential does the tool have for FinOps connectivity to Other Frameworks (e.g., TCO, ITSM, ITIL, or multi-cloud in future phases)?	5

Table 5.8: FinOps Foundation Representative evaluation summary.

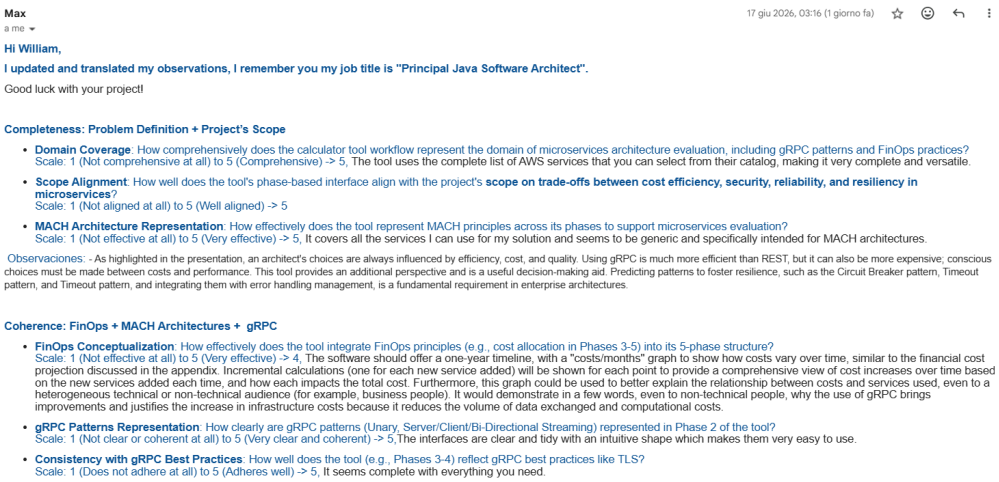
5.0.7 Evaluation results by a Principal Java Software Architect

Massimo Montefusco has more than 20 years of experience in the software industry, and his experience in software architecture was a perfect match for the type of users for the cost-efficiency calculator tool. I contacted him via LinkedIn. The following table shows his evaluation results and observations.

Principal Java Software Architect Evaluation Summary

Criteria	Details	Score
Completeness: Problem Definition + Project's Scope		
Domain Coverage	The tool uses the complete list of AWS services that you can select from their catalog, making it very complete and versatile.	5
Scope Alignment	Aligned with project's scope on trade-offs between cost efficiency, security, reliability, and resiliency.	5
MACH Representation	Covers all services needed for the solution; seems generic and specifically intended for MACH architectures.	5
Observaciones	As highlighted in the presentation, an architect's choices are always influenced by efficiency, cost, and quality. Using gRPC is much more efficient than REST... predicting patterns to foster resilience is a fundamental requirement.	N/A
Coherence: FinOps + MACH Architectures + gRPC		
FinOps Conceptualization	The software should offer a one-year timeline with a "costs/months" graph. Incremental calculations would show how costs vary and justify gRPC infrastructure costs.	4
gRPC Patterns	The interfaces are clear and tidy with an intuitive shape which makes them very easy to use.	5
gRPC Best Practices	Seems complete with everything you need.	5
Perceived Usefulness		
Ease of Understanding	Easy to understand and apply.	5
Decision Support	Effective for tradeoffs in cost, security, reliability, and resiliency.	5
Observaciones	A good FinOps culture is essential for assessing management costs and avoiding unexpected costs.	N/A
Consistency		
Consistency Across Models	Classes comply with technical documentation. Using Record instead of Java Pojo classes is a good choice.	5
Industry Standards	Aligns very well; applications are more efficient and cheaper if they communicate with gRPC.	5
Scalability	The project structure lends itself well to future evolutions.	4
Tradeoff Framework	All the parameters that I can use to define an architecture are present.	5
Extensibility	Only multi-cloud is supported, but external environments (TCO, ITSM, ITIL) can be added in a second version.	4
Observations	The diagrams present the proposal in a clear and modern format. The model is useful and provides significant added value.	N/A

Table 5.9: Principal Java Software Architect evaluation summary.



Max 17 giu 2026, 03:16 (1 giorno fa)

a me

Hi William,

I updated and translated my observations, I remember you my job title is "Principal Java Software Architect".

Good luck with your project!

Completeness: Problem Definition + Project's Scope

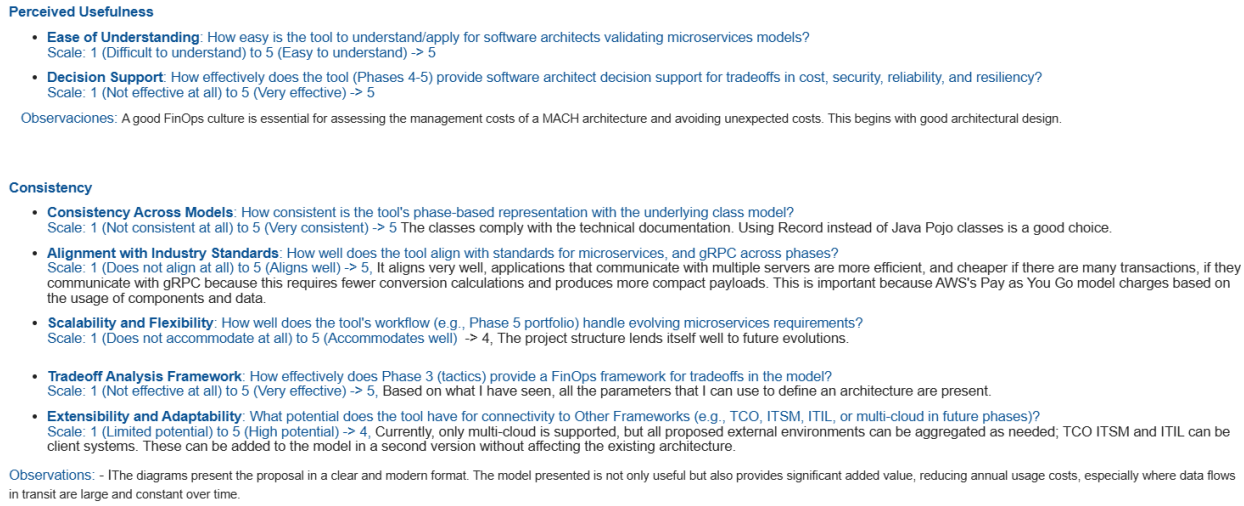
- **Domain Coverage**: How comprehensively does the calculator tool workflow represent the domain of microservices architecture evaluation, including gRPC patterns and FinOps practices?
  Scale: 1 (Not comprehensive at all) to 5 (Comprehensive) -> 5, The tool uses the complete list of AWS services that you can select from their catalog, making it very complete and versatile.
- **Scope Alignment**: How well does the tool's phase-based interface align with the project's **scope on trade-offs between cost efficiency, security, reliability, and resiliency in microservices**?
  Scale: 1 (Not aligned at all) to 5 (Well aligned) -> 5
- **MACH Architecture Representation**: How effectively does the tool represent MACH principles across its phases to support microservices evaluation?
  Scale: 1 (Not effective at all) to 5 (Very effective) -> 5, It covers all the services I can use for my solution and seems to be generic and specifically intended for MACH architectures.

Observaciones: - As highlighted in the presentation, an architect's choices are always influenced by efficiency, cost, and quality. Using gRPC is much more efficient than REST, but it can also be more expensive; conscious choices must be made between costs and performance. This tool provides an additional perspective and is a useful decision-making aid. Predicting patterns to foster resilience, such as the Circuit Breaker pattern, Timeout pattern, and Timeout pattern, and integrating them with error handling management, is a fundamental requirement in enterprise architectures.

Coherence: FinOps + MACH Architectures + gRPC

- **FinOps Conceptualization**: How effectively does the tool integrate FinOps principles (e.g., cost allocation in Phases 3-5) into its 5-phase structure?
  Scale: 1 (Not effective at all) to 5 (Very effective) -> 4, The software should offer a one-year timeline, with a "costs/months" graph to show how costs vary over time, similar to the financial cost projection discussed in the appendix. Incremental calculations (one for each new service added) will be shown for each point to provide a comprehensive view of cost increases over time based on the new services added each time, and how each impacts the total cost. Furthermore, this graph could be used to better explain the relationship between costs and services used, even to a heterogeneous technical or non-technical audience (for example, business people). It would demonstrate in a few words, even to non-technical people, why the use of gRPC brings improvements and justifies the increase in infrastructure costs because it reduces the volume of data exchanged and computational costs.
- **gRPC Patterns Representation**: How clearly are gRPC patterns (Unary, Server/Client/Bi-Directional Streaming) represented in Phase 2 of the tool?
  Scale: 1 (Not clear or coherent at all) to 5 (Very clear and coherent) -> 5,The interfaces are clear and tidy with an intuitive shape which makes them very easy to use.
- **Consistency with gRPC Best Practices**: How well does the tool (e.g., Phases 3-4) reflect gRPC best practices like TLS?
  Scale: 1 (Does not adhere at all) to 5 (Adheres well) -> 5, It seems complete with everything you need.

Figure 5.7: Principal Java Architect Feedback part I.



Perceived Usefulness

- **Ease of Understanding**: How easy is the tool to understand/apply for software architects validating microservices models?
  Scale: 1 (Difficult to understand) to 5 (Easy to understand) -> 5
- **Decision Support**: How effectively does the tool (Phases 4-5) provide software architect decision support for tradeoffs in cost, security, reliability, and resiliency?
  Scale: 1 (Not effective at all) to 5 (Very effective) -> 5

Observaciones: A good FinOps culture is essential for assessing the management costs of a MACH architecture and avoiding unexpected costs. This begins with good architectural design.

Consistency

- **Consistency Across Models**: How consistent is the tool's phase-based representation with the underlying class model?
  Scale: 1 (Not consistent at all) to 5 (Very consistent) -> 5 The classes comply with the technical documentation. Using Record instead of Java Pojo classes is a good choice.
- **Alignment with Industry Standards**: How well does the tool align with standards for microservices, and gRPC across phases?
  Scale: 1 (Does not align at all) to 5 (Aligns well) -> 5, It aligns very well, applications that communicate with multiple servers are more efficient, and cheaper if there are many transactions, if they communicate with gRPC because this requires fewer conversion calculations and produces more compact payloads. This is important because AWS's Pay as You Go model charges based on the usage of components and data.
- **Scalability and Flexibility**: How well does the tool's workflow (e.g., Phase 5 portfolio) handle evolving microservices requirements?
  Scale: 1 (Does not accommodate at all) to 5 (Accommodates well) -> 4, The project structure lends itself well to future evolutions.
- **Tradeoff Analysis Framework**: How effectively does Phase 3 (tactics) provide a FinOps framework for tradeoffs in the model?
  Scale: 1 (Not effective at all) to 5 (Very effective) -> 5, Based on what I have seen, all the parameters that I can use to define an architecture are present.
- **Extensibility and Adaptability**: What potential does the tool have for connectivity to Other Frameworks (e.g., TCO, ITSM, ITIL, or multi-cloud in future phases)?
  Scale: 1 (Limited potential) to 5 (High potential) -> 4, Currently, only multi-cloud is supported, but all proposed external environments can be aggregated as needed; TCO ITSM and ITIL can be client systems. These can be added to the model in a second version without affecting the existing architecture.

Observations: - IThe diagrams present the proposal in a clear and modern format. The model presented is not only useful but also provides significant added value, reducing annual usage costs, especially where data flows in transit are large and constant over time.

Figure 5.8: Principal Java Software Architect Feedback part II.

5.0.8 Summary and analysis of evaluation results

Finally, as a result of this evaluation process, I present the following table in which I summarize the thesis problem's questions I proposed to answer and how the stakeholders perceived the model and the tool in response to those questions.

Thesis Evaluation Synthesis: Thesis Questions vs. Stakeholder Evaluation

Thesis Problem Question	Stakeholder Mapping, Feedback Synthesis, and Evidentiary Evaluation
Mapping Tactics & FinOps Operational Tactics	
How can resiliency, security, and reliability tactics for gRPC microservices map to cloud architecture and integrate with FinOps to promote cost efficiency?	**Architects & Tech Leads:** Validated with high *Scope Alignment* scores (4/5 and 5/5). The model effectively bridges the gap between cross-cutting technical concerns and cost impact. **FinOps Foundation Representative:** Scored *Scope Alignment* 4/5 and *Tradeoff Analysis Framework* 4/5. Validated that the model provides cost, reliability, and resilience vectors, but noted: *"I can see some FinOps Capabilities, but I don't think it reaches the entire framework."* Addressed the scope boundary by stating: *"As far as I saw, It's pretty aligned with the scope, but I cannot comment on the security aspect."* **Principal Java Architect:** Awarded 5/5. Emphasized that predicting patterns to foster resilience is a fundamental requirement, stating: *"Using gRPC is much more efficient than REST... applications are more efficient and cheaper if they communicate with gRPC."*
End-to-End Cost Model & Deployment Variants	
What is the process for creating the model, and what are the key inputs/outputs taking into account deployment variants?	**Tool Survey Insights:** Confirmed that *"Evidence of a formula for cost calculation is presented"* (Score: 5/5). For future iterations, stakeholders noted that inputs must account for multi-region vs. same-region deployment variants and service interconnection (IP) variations. **FinOps Foundation Representative:** Awarded perfect scores (5/5) for both *FinOps Conceptualization* and *Extensibility and Adaptability*. Explicitly validated the model's core mapping by noting: *"It seems perfectly aligned with this topic."* **Principal Java Architect:** Proposed a key output enhancement: *"The software should offer a one-year timeline with a 'costs/months' graph"* to visually capture incremental calculation variances (Score: 4/5).
Practical Tool for Software Architects	
How can the proposed model be developed into a convenient, real-world tool for software architects working in cloud environments?	**Principal Java Architect:** Highly praised the technical consistency (Score: 5/5), validating that utilizing the complete AWS service catalog makes the framework exceptionally versatile. Noted that building the backend using Java `Record` types instead of standard POJOs ensures excellent maintainability and scalability (Score: 4/5). **FinOps Foundation Representative:** Identified a clear gap in user experience, scoring *Ease of Understanding* at 3/5, and observed: *"I think that the UI need to be improved. Maybe a more friendly UI or a more simple one. Even adding some notes that could guide the user."* **Tool Survey Evaluation:** Noted that adding a live telemetry use case (e.g., analyzing traces, logs, and metrics in Prometheus) would make the tool highly effective for operational architecture environments (Score: 5/5).
Perceived Usefulness & MACH Expectations	
How can the tool's perceived usefulness be aligned with architects' expectations for calculating cost-efficiency in MACH architectures?	**Architecture Evaluation Survey:** Perfect scores for *Scope Alignment* (5/5) and *MACH Representation* (5/5). Feedback explicitly stated: *"The model was really impressive and left us talking about why this functionality didn't already exist. It would be incredible to see something like this available."* **FinOps Foundation Representative:** Provided a crucial counter-perspective on tool adoption, scoring *Practical Relevance* at 3/5 and stating: *"I still don't see a practical usability. I dunno if it's a lack of knowledge on my end, but I still think someone can perform the same activity with a simple automation process. This could impact the growth and usage of this tool."* Conversely, awarded 5/5 for *MACH Architecture Representation* and 4/5 for *Alignment with Industry Standards*. **Tech Lead Manager:** Stated the tool provides an entirely fresh and alternative outlook on resource optimization, proving highly useful when setting up new cloud infrastructure. **Principal Java Architect:** Rated *Decision Support* at 5/5, highlighting that a solid FinOps culture is mandatory for architects to prevent unexpected costs.

Table 5.10: Thesis evaluation synthesis: Thesis questions vs. Stakeholders evaluation.

5.1 Chapter Summary

In summary, this chapter provided different key points of view about the end-to-end model proposed and the cost-efficiency calculator tool for software architects that was designed to validate the model, its inputs, and outputs. Most of the observations were considered for inclusion before the final version of this thesis, while others were included as future work in the next section.

Chapter 6

Conclusions

"Why do we do this? We do it because it is fun when it's not infinitely frustrating. We do it because it's worth doing. We do it because people are depending on us. But we also do it because we owe it to the next bunch of 10 year olds."

Russ Olsen, "To The Moon" conference at GOTO Copenhagen 2024 (Olsen, 2024)

In this chapter, I present the conclusions I reached after proposing and implementing the End-to-end model for evaluating the cost-efficiency of MACH Architectures that use gRPC as their interaction style, considering the new paradigm for calculating the Total Cost of Ownership in those architectures (TCC and TCO).

6.1 Conclusions

The main motivation for doing this thesis was to contribute to the software engineering community with a model and a tool that helps us reduce the gap we usually have in financially justifying our technical decisions, and in consequence that also allows us to become **Frugal Architects** who promote c**ost-efficiency** and **sustainability** in software architectures, MACH ones in particular.

The End-to-end model allows software engineers and architects to not only be aware of the **Total Cost of Change (TCC, or marginal cost)** of every architectural decision they make for their MACH Architectures but also the whole **TCO cost-efficiency estimation** for a month and a year.

Furthermore, the model enables the evaluation of the impact of **FinOps optimization strategies** on a **cloud-native software architecture**. That helps software engineers and architects envision, in advance, how the costs might behave.

Moreover, by including FinOps principles in advance and during the Inform phase, software engineers and architects have the capacity to engage in financial discussions about the unit economics that provide visibility into the cost and profitability of their cloud native software solutions in this context. Therefore, the model works not only as a logical model for the cost-efficiency calcula-

tor tool but also as an integrator between the gRPC Interaction Style, software architecture, and the AWS Well Architected Framework pillars involved in cost-efficiency, and the FinOps operating model.

On the other hand, based on the cost-efficiency calculator tool, it is evident that it provides a map to navigate through the operative process defined for the model, so that software engineers and architects can be formally guided through establishing budgets and forecasting that allows them to be more financially conscious of their architectural decisions. It is not only about how many cloud services or architectural tactics and patterns we, software engineers and architects need to include, but also about how our decisions impact costs in the long term, and how they affect revenue.

Moreover, by including the possibility of comparing cost and revenue unit economics for the services calculated in the portfolio for a complete MACH Architecture, we gain valuable insights for any software engineer, architect, and leader in the cloud-native era. With this model in mind, those industry roles will find it easier to adapt to the transition and consolidation of cloud environments.

Finally, the impact of not adopting FinOps is that *"when you're focused on speed and innovation, it's easy for cloud bills to soar. In response, companies too often clamp down on cloud, causing innovation to slow and threatening to make them less competitive in the process"* (Storment & Fuller, 2023). Now we count on a model and a tool to prevent those cloud bills from soaring without sacrificing speed and innovation. Therefore, it is an honour to conclude that as software engineers and architects, we will be more confident in improving the trustworthiness in our relationship with finance departments, and we will be more financially responsible and ethical to the community and the institutions we belong to.

6.2 Future Work

As Montalion (2024) says about technology design: *"We build what we think. We structure thinking through communication, and then we craft it into something actionable"*. I had not thought of this thesis as an isolated project. I conceived it as a second step of a project I defined in 2016, the year in which I proposed my Telematics Engineering thesis on evaluating resource utilization while avoiding idle resources in computations that require processing vast volumes of data, for which Apache Mesos (The Apache Software Foundation, 2012) was very useful, even though nowadays Kubernetes (The Kubernetes Authors, 2026) has won the industrial adoption race between them. That established a base from a computational resource management perspective, but not from a financial one. This thesis satisfies that part with its end-to-end model for software architectures in my operational context of interest: e-commerce. However, *"Technology skills (alone) are not enough. We don't think in systems. To move from software to systems, we need to think differently (together)"* (Montalion, 2024), and in order to build that whole system, it might require more phases, more projects, and more time, following the Duomo di Milano's example of becoming a symbol of beauty that sings to the heart, the mind, and the soul of everyone who visits it (Veneranda Fabrica del Duomo di Milano, 2023). Therefore, future work is defined as an integration between both theses by also covering the observations given by the evaluators and the initiatives that were not part of the scope of this thesis. In consequence, the following are the main objectives for future

work:

1. **Include Future Work from the Conceptual Design**: in the conceptual design section, some conceptual mappings were highlighted in red, meaning that those need to be part of the future work for this project in order to cover all the frameworks key components in the cost-efficiency calculator, especially the ones related to containerization for cloud native deployments, in which tools like Karpenter (Amazon Web Services, 2026i) and Quick Suite (Amazon Web Services, 2025s), and the OpenCost specification (Ray, 2023) will be needed.
2. **Include cloud pricing values in the cost factors and support multiple cost criteria for each cost factor**: in the cost-efficiency calculator proof of concept tool, I retrieved the cloud service cost directly from the AWS API, obtained the pricing value for each type of service, and displayed it in the Thymeleaf UI through the controllers. That helped calculate the pricing values in the JavaScript scripts associated with the Thymeleaf template before sending them to the TCO and Unit Economics controller so that it performs the final unit economics and TCO calculations. In the current version of the proof of concept tool, the pricing values are not associated directly as values in the CostFactor class, and it remains only informational to determine whether it is a networking cost or an infrastructure cost. In the future, the CostFactor class should include the total pricing value as well to improve the TCO and unit economics calculation completely from the backend services, by looking through the cost factors of each architectural decision once they are selected and set by the software architect. Finally, as some cloud services might present different pricing values based on various criteria, such as the Amazon CloudWatch service, which has pricing for data ingestion and data archival Amazon Web Services (2026a), **multiple cost criteria** can be supported individually in the future for a single cost factor. The current proof of concept tool computes those criteria in the Thymeleaf UI in its JavaScript scripts.
3. **Sustainable software systems integration**: integrate both the Telematics Engineering thesis and the Master's Degree in software Engineering thesis in order to propose an integrated model for sustainable technological platforms in evaluating solution architectures that promote resource utilization, cost-efficiency, and sustainability (Eldh, 2026). For sustainability, I have already certified the **GreenOps** Philosopher course (Credly, 2026), which positions GreenOps strategies as enablers of cost-efficiency initiatives when they conflict with the same priority. That will need deeper study not only in GreenOps but also in green software architectural patterns and tactics (Joint Development Foundation Projects, 2026a, 2026b).
4. **Automation of MACH Architectures and FinOps practices using Infrastructure as Code (IaC)**: following the observations provided by the FinOps Foundation representative regarding having defined as an outcome of this cost-efficiency calculator the code to deploy the infrastructure that supports the MACH Architecture portfolio, including the FinOps allocation strategy for granting the FinOps Inform phase in an operative cloud environment. The benefits of having this defined as code are that it improves the *"traditional processes and techniques for changing infrastructure safely"* (Morris, 2020) that *"are not designed to cope with a rapid pace of change"* (Morris, 2020), and that *"throttle the benefits of modern Cloud Age technologies—*

slowing work down and harming stability" (Morris, 2020). This way, software architects would be able to make an initial validation of how the MACH Architecture's infrastructure with the improvements behaves initially and reduce the gap between the cost-efficiency estimation and the real operative cost to tune the model.

5. **Design an AI System with a custom LLM to simplify complex TCC and TCO calculations and the building of the MACH architecture**: while evolving the End-to-end model and the Cost-efficiency calculator in the future, some variances, calculations, and FinOps strategy suggestions might be more complex than those included in the scope of this thesis. Therefore, an AI system might need to be design by following the pillars and principles of fairness, trustworthiness, safety, ethics, privacy, and transparency defined by Len Bass (Bass, 2025), along with a custom Language Model (Raschka, 2025) that processes pre-defined prompt-based TCC and TCO calculation requests. In this way, the end-to-end model can be extended to align with the emergence of Generative AI to create prompts for developing the MACH composable architecture, as envisioned by Solomonson (2024).
6. **Adapt the current gRPC patterns to the current state of the art with the QUIC protocol over HTTP/3**: the community is currently working on making **gRPC** support HTTP/3 (Brandhorst-Satzkorn, 2019; Newton-King, 2022) by using the QUIC protocol as intended by the Cronet project (Mastrangelo, 2018).
7. **Using the Residues approach to generate the MACH Architecture**: stressing the MACH Architecture, as suggested by Barry O'Reilly: *"A random simulation of stress is a better way of generating a software architecture than prediction, requirements analysis, reuse of patterns, or reactive change management by coding."* (O'Reilly, 2024). This approach will allow us to continuously improve the model based on the behaviour of deployed stressed scenarios.
8. **Multi-cloud approach and FOCUS**: to extend the model to support multi-cloud MACH Architectures in which multiple cloud providers are contemplated to offer the best-of-breed components, and that allow us to use FinOps Foundation Project a Series of LF Projects (2024e) specifications to manage TCC and TCO reports.
9. **Improve the UI/UX of the cost-efficiency calculator tool**: as suggested by the FinOps Foundation representative in his evaluation, another improvement needed for a future work is to promote the usability of the cost-efficiency calculator tool to grant a better user experience (UX). Finally, another important improvement as the calculator capabilities increments is using modern frameworks to decoupled the frontend from the backend side, and using microfrontends as an architectural style to promote independent deployability, technology heterogeneity, and reusability through modularity of the UI components (Mezzalira, 2025).
10. **Support of multiple ProtocolBuffer files** that include different structures from the ones defined for the business use cases of this project for the four gRPC communication patterns.

6.3 Lessons learned

Project planning and execution were difficult to manage along the way, exactly as I thought the first time I wrote the thesis proposal. Taking into account that while working on this thesis, I was

actively employed in an international company where a heavy load of responsibilities was placed on my role, it was a real challenge to keep myself energetic, motivated, and hopeful during times when I felt frustrated and tempted to resign. I worked on my fortitude to move forward and also sometimes waited for the right moment to have a love connection with the thesis, as only through love and service do we reborn.

Moreover, the model evaluated in KR1 of Objective 5 in the Evaluation chapter was transformed during the development of the tool. As a result of the refactoring and the possibility of adapting it to the model itself, some aggregations and parts that were initially conceived but were not necessary were removed, such as a persistence model that used to accompany the logic one since no item needed to be persistent, pricing calculation logic via a builder schema, and finally, a representation in percentages of the impact of selecting quality attributes against costs, as pricing numbers were used instead in this project.

Another difficulty I had was clearly defining an integration between the MACH Mindset and two large technological frameworks (FinOps and the AWS Well-Architected Framework) through a simple conceptual model to design an end-to-end model that satisfies the thesis main goal. The amount of detail and complexity led me to make an extra effort to dive deeply into each of those topics while establishing conceptual linkages among them. As a result, in this project, I had the opportunity to obtain the following badges from official certifications that support the Reference Framework I used as the pillars for this thesis: gRPC [Java] Master Class (Credly, 2024b) from the O'Reilly course (O'Reilly, 2019), Distributed Data Patterns for Microservices (Chris Richardson Consulting, 2025), and all the rideMACH certifications for MACH Architectures, the most relevant one being Intro to Total Cost of Ownership (TCO) | rideMACH (Credly, 2025), AWS Certified Cloud Practitioner (Credly, 2023), FinOps Certified Engineer from the FinOps Foundation (Credly, 2024a), FinOps for Containers from the Linux Foundation (The Linux Foundation, 2024), and the Professional Diploma in Unit Economics Management by the MTF Institute of Management and Finance (2024).

Moreover, my thesis is an integrative project of the following Master's Degree program's subjects and the complementary university diplomas I studied in Pontificia Universidad Javeriana Cali, which I completed during this period: Ethics and Impacts in Technology, Design Patterns and Strategies, Integration Architectures, Cloud Architecture, Software Architecture, Software Testing, Software Project Management and Quality, Diploma in DevOps, Diploma in Financial Tools for Decision-Making, and Diploma in E-commerce.

Furthermore, I had the opportunity to apply the knowledge acquired in the **GOTO Copenhagen 2024 Masterclass, Visualizing Software Architecture with the C4 Model, with Simon Brown** (Trifork, 2024), in which Structurizr was introduced. Along with the use of AI tools like Claude Code and Google Gemini, I developed and modeled the software architecture views effectively and efficiently by addressing all the concerns and architectural drivers and guiding the AI in the details of my implementation. I was also able to improve my communication skills in presenting those software engineering and architecture diagrams by following the suggestions and tips found in Jacqui Read's book about what she defines as *"virtual literacy, the ability to understand and create visuals"* (Read, 2023); considering that *"visual elements in software architecture*

and design communicate key information" (Read, 2023). As she says: *"Successful communication is the art and science of sharing or exchanging ideas and information, using a common set of symbols, signs, or behaviours, resulting in shared understanding"* (Read, 2023), including technical and non technical audiences like finance. I think I have learned that crucial lesson for my professional career, which is to be sustained in the future so that my communication gap with architects, business and finance, and the rest of the FinOps personas is reduced over time.

Moreover, arriving at the definition of this end-to-end model was a complex task that required a large number of iterations and validations. Even though I relied on the help of Claude Code (Anthropic, 2026), Gemini, and Perplexity AI as tools to guide the development and validate the cost-efficiency calculator for Software Architects, the difficulty of refactoring, validating, and testing against the model to ensure consistency with the FinOps, MACH, and AWS mapping was extremely challenging; it was also entertaining and enriching.

Finally, this is the last lesson I learned from this project: to truly and always believe in my capacity, my discipline, and my mindset to sacrifice myself, hoping *"that something better might be attained in the future by giving up something of value in the present"* (Peterson, 2019), and to make the necessary effort to face difficult challenges that are worthwhile, meaningful, and right, with *"the lively wit, the firm will, the iron tenacity, all that accumulation of circumstances that we call fortune"* (Serao, 2025), as a way to acquire a countercultural mindset and attitude, which allows me to extend to my community and my country the invitation made by Russ Olsen in the GOTO Copenhagen conference in 2024: *"we chose to go to the moon not because it's easy, but because it's hard [...] and when you get home, from the bottom of my heart, I hope that you choose to do something hard"* (Olsen, 2024).

Chapter 7

Annexes

7.1 RESTful Architectural Style versus gRPC Interaction Style

From a MACH perspective, microservices can implement different APIs for communicating among themselves. Some of the most popular options are RESTful APIs and gRPC APIs. The following is a table comparing both from an architectural perspective so that a Software Architect has enough criteria to decide between them.

Architectural Comparison Matrix: REST APIs vs. gRPC

Criteria	REST APIs (Architectural Style)	gRPC (RPC Interaction Style)
Paradigm	Resource-oriented architectural style.	Action-oriented remote procedure calls.
Use Case	Web architectures and public-facing CRUD APIs.	High-performance microservices and streaming.
Gateway	Native cloud support (e.g., AWS API Gateway).	Requires HTTP/2 / gRPC proxies or sidecars.
Caching	Cacheable by default via GET methods (RFC 9111).	Dynamic, non-cacheable pipeline layer traffic.
Resiliency	Handled via application-level patterns (Retries).	Native protocol-level tactics (Timeout, Retry).
IDL / Doc	Human-readable OpenAPI / Swagger specifications.	Machine-readable Protocol Buffers specifications.
Load Bal.	Native standard L4/L7 routing (AWS ALB).	Supported via AWS ALB using end-to-end HTTP/2.
HTTP	Typically HTTP/1.1 (plain-text, sequential TCP).	Strictly HTTP/2 (multiplexed single TCP stream).
Payload	Plain text (JSON, XML profiles).	Binary serialization (Protocol Buffers formatting).
Size	Verbose text string footprint overhead.	Highly compact (typically half the size of JSON).
Streaming	Unidirectional Request-Response patterns only.	Unary, Client, Server, and Bi-directional.
Execution	Explicit resource endpoints and HTTP verbs.	Local client stub methods called directly from IDL.
Security	Standard profiles (OAuth 2.0, TLS infrastructure).	Strong transport security (TLS/mTLS interceptors).

Table 7.1: Comparison between REST APIs and gRPC

7.2 Additional conceptual mappings

Developing an **end-to-end model** involves not only abstract concepts but also a clear explanation of how these ideas can be turned into practical implementations. This chapter serves as the

crucial bridge, moving from the theoretical blueprint presented earlier to the concrete reality of the proposed system.

In this annex, I will explain the alignment of the proposed model and its implementation approach with the principles and good practices outlined in both the **AWS Well-Architected Framework** (Amazon Web Services, 2023a) and the **FinOps Framework** (FinOps Foundation Project a Series of LF Projects, 2025h).

7.3 Mapping Conceptualization with the FinOps Foundation Framework and the AWS Well-Architected Framework to operate the End-to-end model

To effectively operate the proposed end-to-end model, it is essential to contextualize its design and implementation within established industry standards. This section undertakes a detailed mapping exercise, correlating not only the conceptual underpinnings of our model but also specific components of the end-to-end process and the methodologies followed by the Software Architect with the principles and pillars of the FinOps Foundation Framework (FinOps Foundation Project a Series of LF Projects, 2025h) and the AWS Well-Architected Framework (Amazon Web Services, 2023a) (Amazon Web Services, 2024a) (Amazon Web Services, 2024b) (Amazon Web Services, 2025v)(Amazon Web Services, 2019).

7.3.1 FinOps Framework Mapping

As mentioned in Storment and Fuller (2023), *"every organization adopting the cloud will need to develop mechanisms to manage cloud cost,"* therefore, *"partnering with Engineers to Enable FinOps* is crucial to *instill a culture of FinOps by building up engineering motivation to be cost-efficient rather than dictating a mandate"*. Hence, *"FinOps Practitioners need to work with engineers [...] to make cost efficiency as big a consideration as security, performance, reliability, or sustainability. Focusing on only one measure of success is not an option"* (Storment & Fuller, 2023). As a result, the following are some of the **Principles for Enabling Cost-Efficient Engineering** that are relevant to the model:

Maximize value rather than reduce costs: this means that the FinOps value is not about reducing costs only but instead that the cost the software architect is incurring with the MACH Architecture is justified by the value they provide (the ingress that grants the ARPU is greater than the CAC unit economics, which in our case is represented by the Cost per user). The following formula is presented as a point of reference to evaluate the quality from FinOps perspective.

$$\textbf{Quality} = \frac{\text{results of work efforts}}{\text{total costs}}$$

Results of work efforts refer to the measurable outputs, outcomes, or business value generated by an organization's activities—specifically, those efforts related to cloud financial operations and management.These results are typically tracked using a combination of metrics and key performance indicators (KPIs) that reflect both technical and business value. Examples include: Percentage of

workloads running at optimal utilization, Reduction in cloud spend without loss of performance, Number of new features delivered per dollar spent, Business revenue or customer engagement attributable to cloud-enabled services.

As shown in the following fragment of the class diagram, the model promotes this principle by allowing the Software Architect to choose not only the Cloud Service that promotes the Architectural Pattern or Tactics but also the cloud service that supports its implementation on the cloud and the FinOps strategy that is needed to ensure cost efficiency in implementing those cloud services.

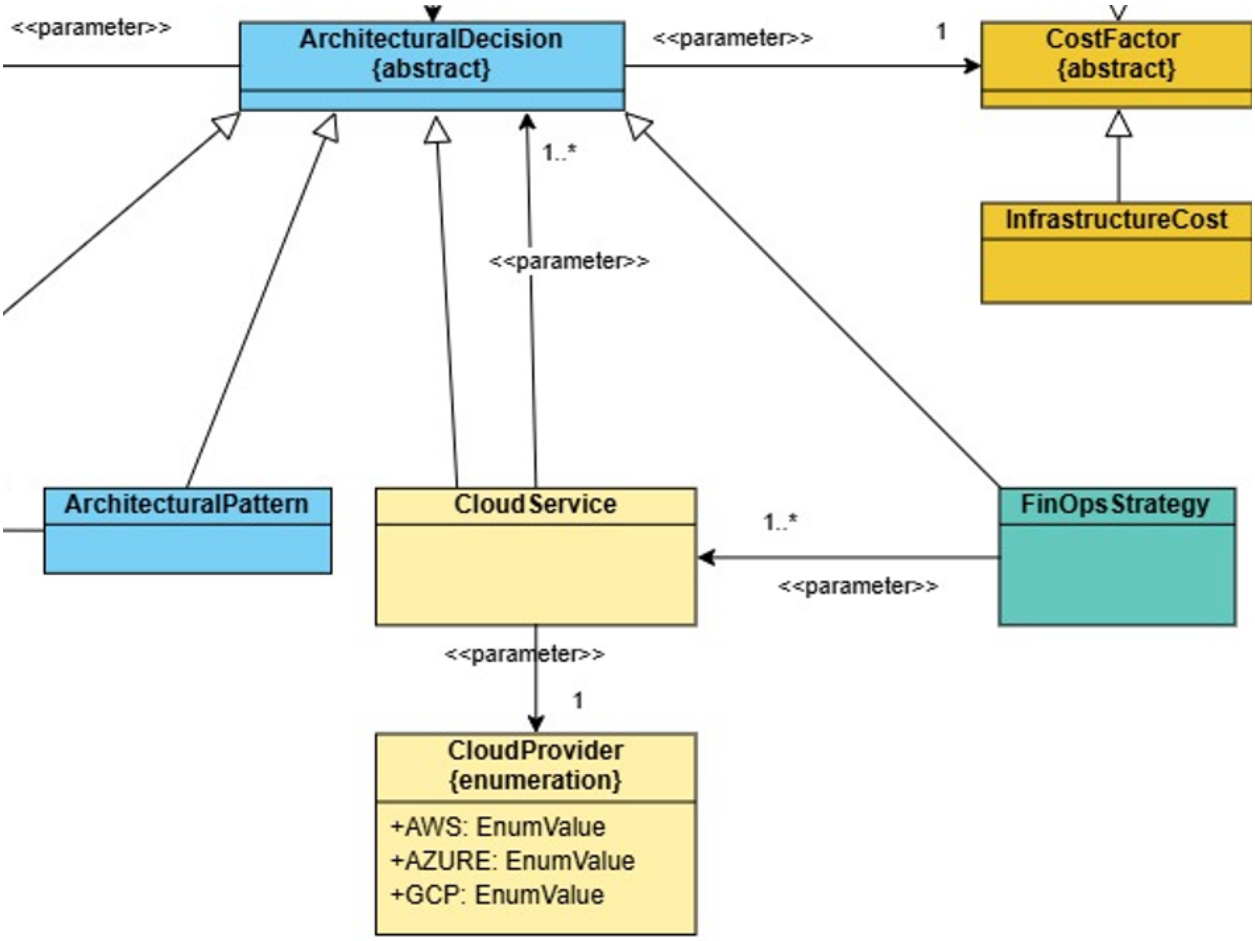


Figure 7.1: End-to-end model mapped to FinOps framework through FinOps strategy applied to cloud services.

7.3.1.1 Prioritize improving communication by using the FinOps Terminology

FinOps Foundation Project a Series of LF Projects (2024c) defines the FinOps terminology to be used consistently, as it was a ubiquitous language for bridging the gap between the different FinOps Personas. Terms like budget, forecast, unit economics, cost-awareness, etc, were used in

the conceptual mapping presented in the Project Development section.

7.3.1.2 Introduce Financial Constraints Early in the Product Development

The end-to-end model also maps what Storment and Fuller (2023) says about considering the cost of re-architecting. This is evident in the process software architects should follow to generate the **detailed budget and the financial forecast** of implementing the different software tactics and patterns, the cloud services that support them, and the FinOps configuration over those services in their solutions. First, after mapping the cloud services to implement the tactics and patterns chosen by the software architect, the **base cost (or cloud service cost without cost optimization strategies)** is calculated by the model and presented as an outcome to the software architect (Activity 4).

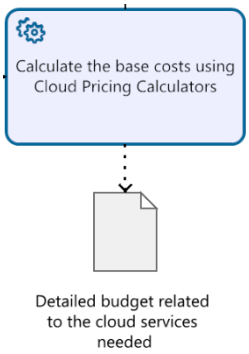


Figure 7.2: End2EndModel-FinOps Principle-Introduce Financial Constraints Early in the Product Development-Part I.

Then, the Software Architect needs to choose the cost optimization strategies (FinOps strategies) to be implemented, and then another budget is calculated along with a financial forecast based on the initial base cost and the commitment of the optimization strategies (Activity 7).

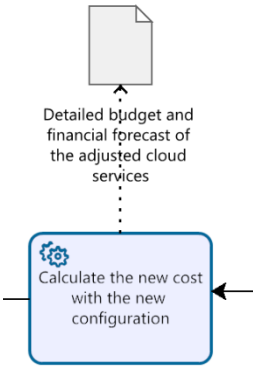


Figure 7.3: End-to-end Model-FinOps Principle-Introduce Financial Constraints Early in the Product Development-Part II.

Therefore, the partnership between FinOps and Engineering teams could be established *"by helping engineers design cloud services to be more efficient and work on tools that avoid creating waste up front"* (Storment & Fuller, 2023). This is the one covered by the end-to-end model, as it is expected for **FinOps practitioners** to help software architects define cost optimization strategies, as shown in the following activities 5 and 6 of the operative process:

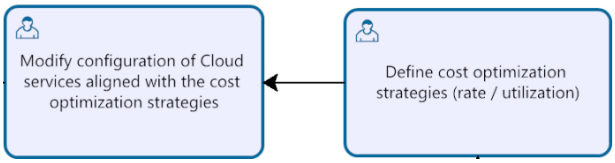


Figure 7.4: End-to-end Model-FinOps Partnership Model with Software Architects and Engineers.

Storment and Fuller (2023) defines an **operating model** that combines FinOps practices with Cloud Cost Management practices as a list of activities that engineers use in their practice as needed. According to the **FinOps Personas** presented by Storment and Fuller (2023), the **Engineering Team** to which **software architects** belong, needs to know how each of their tasks contributes to a particular capability and the FinOps Framework . Besides, each capability has a set of **Functional Activities** (either tasks or processes) to perform actions, and to manage data. The following are the **capabilities** that are covered by the **End-to-end model**, organized by FinOps Phases.

1. **Domain**: **Understanding Cloud Usage and Cost**. **Desired Outcome**: Drives Accountability and Transparency. **Reason**: The end-to-end model proposes a way of calculating costs that depends on the specification of the FinOps strategies to be used for the cloud services supporting the architectural tactics and patterns to promote architectural drivers defined as quality attributes. This drives Software Architects to use cost allocation tags for each cloud service and analyze the data generated in operations regarding the cost of usage and rate cost for **showback** or **chargeback** purposes. **Capabilities**:

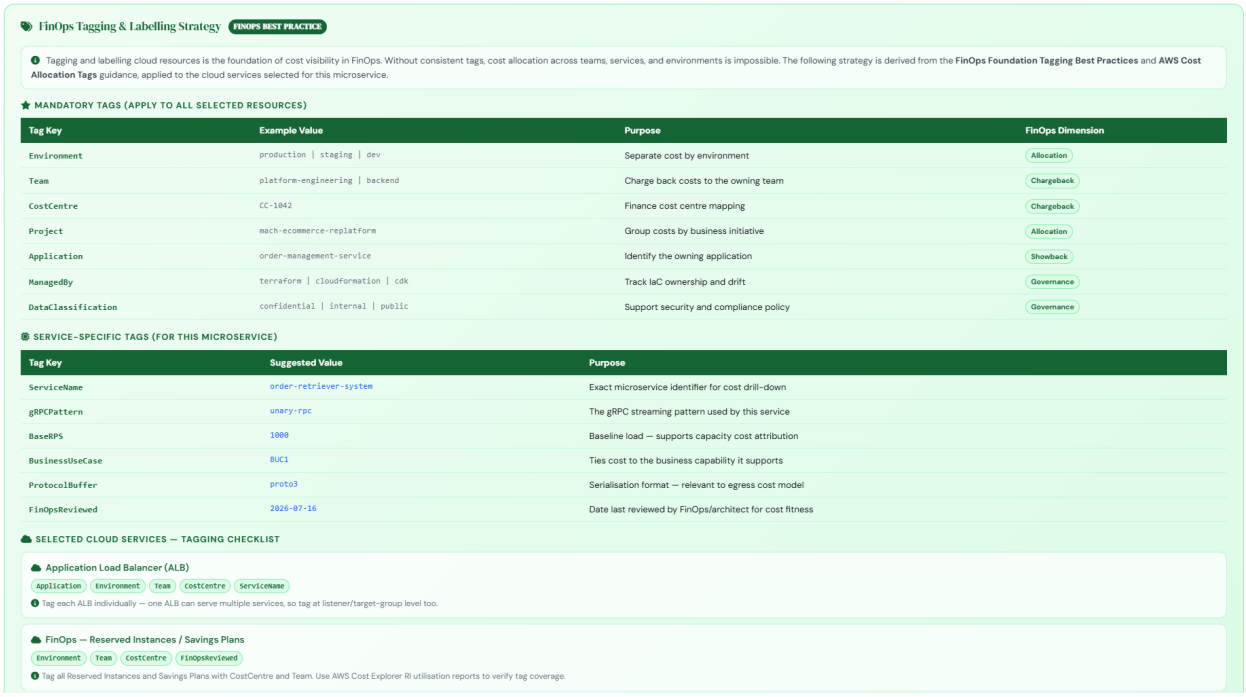
FinOps Domain Mapping

Capabilities	Phase	FinOps Engineer Persona's Functional Activities	Measure of Success
Cost Allocation	Inform	*"Determine how and when metadata will be applied to hierarchical groupings and resources"* aligned with the definition: *"Define specific tags, labels, naming standards, grouping structures used to identify that a cost is in a particular grouping"* (FinOps Foundation Project a Series of LF Projects, 2025k). *"Identify and provide all metadata sources required for analysis and cost allocation"* (FinOps Foundation Project a Series of LF Projects, 2025k).	*"The majority of cloud costs can be categorized and allocated directly to an organizational unit."* (Storment & Fuller, 2023)

Table 7.2: FinOps Capabilities, Activities and Phases mapping covered by the End-to-end Model in the FinOps Domain of Understanding Cloud Usage and Cost (FinOps Foundation Project a Series of LF Projects, 2025f)

.

Regarding that capability, the following is the FinOps labelling and tagging strategy automatically generated for the service modelled:



FinOps Tagging & Labelling Strategy — FINOPS BEST PRACTICE

Tagging and labelling cloud resources is the foundation of cost visibility in FinOps. Without consistent tags, cost allocation across teams, services, and environments is impossible. The following strategy is derived from the **FinOps Foundation Tagging Best Practices** and **AWS Cost Allocation Tags** guidance, applied to the cloud services selected for this microservice.

MANDATORY TAGS (APPLY TO ALL SELECTED RESOURCES)

Tag Key	Example Value	Purpose	FinOps Dimension
Environment	production | staging | dev	Separate cost by environment	Allocation
Team	platform-engineering | backend	Charge back costs to the owning team	Chargeback
CostCentre	CC-1042	Finance cost centre mapping	Chargeback
Project	mach-ecommerce-replatform	Group costs by business initiative	Allocation
Application	order-management-service	Identify the owning application	Showback
ManagedBy	terraform | cloudformation | cdk	Track IaC ownership and drift	Governance
DataClassification	confidential | internal | public	Support security and compliance policy	Governance

SERVICE-SPECIFIC TAGS (FOR THIS MICROSERVICE)

Tag Key	Suggested Value	Purpose
ServiceName	order-retriever-system	Exact microservice identifier for cost drill-down
gRPCPattern	unary-rpc	The gRPC streaming pattern used by this service
BaseRPS	1000	Baseline load — supports capacity cost attribution
BusinessUseCase	BUC1	Ties cost to the business capability it supports
ProtocolBuffer	proto3	Serialisation format — relevant to egress cost model
FinOpsReviewed	2026-07-16	Date last reviewed by FinOps/architect for cost fitness

SELECTED CLOUD SERVICES — TAGGING CHECKLIST

Application Load Balancer (ALB)

Application · Environment · Team · CostCentre · ServiceName

Tag each ALB individually — one ALB can serve multiple services, so tag at listener/target-group level too.

FinOps — Reserved Instances / Savings Plans

Environment · Team · CostCentre · FinOpsReviewed

Tag all Reserved Instances and Savings Plans with CostCentre and Team. Use AWS Cost Explorer RI utilisation reports to verify tag coverage.

Figure 7.5: FinOps labelling and tagging strategy automatically generated for the service modelled.

2. **Domain**: **Quantify Business Value (FinOps Foundation Project a Series of LF Projects, 2025g)**. **Desired Outcome**: *"Plan, estimate, and forecast needs for cloud plus intersecting FinOps Scopes using relevant cost, impact, and usage data"* (FinOps Foundation Project a Series of LF Projects, 2025g). **Reason**: The End-to-end model proposes an incremental cost calculation of a software architecture that will then be supported and operated through cloud services, with costs presented as marginal costs as well as the increments at the end of each calculation iteration. This marginal cost is equivalent of the Total Cost of Change applied in the model each time an architectural decision is made. Therefore, the exercise of calculating the costs in the model is itself a forecast of how much the iteration would cost for operating the chosen software architecture in the cloud.

FinOps Domain Mapping — Quantify Business Value

Capabilities	Phase	FinOps Engineer Persona's Functional Activities	Measure of Success
Unit Economics	Optimize	*"Define and document goals the organization is trying to achieve through use of cloud". "Define and document measurements that are appropriate to evaluate our cloud use and cost"* (FinOps Foundation Project a Series of LF Projects, 2025d). *"Define unit economics and KPIs which satisfy the needs of those organizational goals"* (FinOps Foundation Project a Series of LF Projects, 2025d).	*"percentage of teams, personas, or stakeholders that are using unit economics metrics to communicate about cloud use"* (FinOps Foundation Project a Series of LF Projects, 2025d)
Budgeting	Optimize	*"Get approval for planned changes which will have a negative impact to our cloud spend or value projections / budgets using approved processes with budget owners"* (FinOps Foundation Project a Series of LF Projects, 2025b)	*"Budgeting leverages appropriate discount-adjusted, amortized cloud usage data, as identified in the Budgeting Strategy"* (FinOps Foundation Project a Series of LF Projects, 2025b). *"Budget cadence that includes rolling forecasts to update budgets based on business drivers"* (**architectural drivers or quality attributes** in case of Software Architects) (FinOps Foundation Project a Series of LF Projects, 2025b).
Forecasting	Optimize	*"Work with FinOps stakeholders to identify actionable optimization opportunities to avoid forecasted overspend"* (FinOps Foundation Project a Series of LF Projects, 2025b). *"Get approval for planned changes which will have a negative impact to our cloud spend projections / budgets"* (FinOps Foundation Project a Series of LF Projects, 2025b).	*"Forecast models leverage discount-adjusted, amortized cloud usage data"* (FinOps Foundation Project a Series of LF Projects, 2025b). *"Forecast frequency that includes intermediate forecasts to update budgets based on business drivers"* (**architectural drivers or quality attributes** in case of Software Architects) (FinOps Foundation Project a Series of LF Projects, 2025b).

Table 7.3: FinOps Capabilities, Activities and Phases mapping covered by the End-to-end Model in the FinOps Domain of Quantify Business Value (FinOps Foundation Project a Series of LF Projects, 2025g)

.

3. **Domain**: **Optimize Usage and Cost (FinOps Foundation Project a Series of LF Projects, 2025f)**. **Desired Outcome**: *"Analyse and manage spend-based commitments and resource-*

based commitments. Design solutions with ***cost-awareness*** *to maximize business value and achieve performance, scalability, and* ***operational objectives****"* (FinOps Foundation Project a Series of LF Projects, 2025f). **Reason**: that the model includes the FinOps strategies applied to cloud services when those are part of the architectural decisions of the software architect.

FinOps Domain Mapping — Optimize Usage and Cost

Capabilities	Phase	FinOps Engineer Persona's Functional Activities	Measure of Success
Rate Optimization	Optimize	1. *"Collaborate with the FinOps team to provide information on current and future resource use plans, notifying them of changes that will substantially impact my usage, and highlighting important services"* (FinOps Foundation Project a Series of LF Projects, 2025k). 2. *"Understand and incorporate discounted cost data into my reporting and analysis as appropriate in the rate optimization strategy"* (FinOps Foundation Project a Series of LF Projects, 2025k).	1. *"Ability to measure the overall effective savings rate of Cloud Rate Optimization efforts for both technology-based and monetary-based commitment discounts"* (FinOps Foundation Project a Series of LF Projects, 2025k). 2. *"Purchasing of commitment discounts are viewed as investments by stakeholder teams like Execs / Procurement / Finance; the investment is the cost of the commitment over the entire period, and the return is the savings provided"* (FinOps Foundation Project a Series of LF Projects, 2025k).
Architecting for Cloud	Optimize	1. *"Maintain an awareness of cloud services, deployment patterns, development models, across my organization and generally"* (FinOps Foundation Project a Series of LF Projects, 2025k). 2. *"Follow and evaluate my workloads according to published or prescribed architectures"* (FinOps Foundation Project a Series of LF Projects, 2025a)	1. *"Measurement of cost efficiency through infrastructure savings, migration and support costs".* 2. *"Operational resiliency improvement in service quality and security risk posture"* (FinOps Foundation Project a Series of LF Projects, 2025a). 3. *"Enable a culture of rapid experimentation to drive innovation and cloud transformation"* (FinOps Foundation Project a Series of LF Projects, 2025a) (through Infrastructure as Code (Morris, 2020) for the architectural deployment).

Table 7.4: FinOps Capabilities, Activities and Phases mapping covered by the End-to-end Model in the FinOps Domain of Optimize Usage and Cost (FinOps Foundation Project a Series of LF Projects, 2025f)

.

4. **Domain**: **Manage the FinOps Practice (FinOps Foundation Project a Series of LF Projects, 2025e)**. **Desired Outcome**: *"enables continuous improvement to change and align the entire organization – its people, processes, and technology – to adopt FinOps and*

use cloud and other technology in ways that create value for the company" (FinOps Foundation Project a Series of LF Projects, 2025e). **Reason**: The End-to-end model, through the FinOpsStrategy class, has an intersection with the FinOps Framework. In the end, as a final result, the model offers to the Software Architect detailed budget and financial forecast of the adjusted cloud services, which will be presented and integrated along with other financial reports to justify the Software Architecture acquisitions and commitments in front of the Procurement and Finance Teams.

Figure 7.6: Disclaimer inviting the software architect to present his cost-efficiency estimations to the FinOps personas.

FinOps Domain Mapping — Optimize Usage and Cost (Continued)

Capabilities	Phase	FinOps Engineer Persona's Functional Activities	Measure of Success
Rate Optimization	Optimize	1. *"Collaborate with the FinOps team to provide information on current and future resource use plans, notifying them of changes that will substantially impact my usage, and highlighting important services"* (FinOps Foundation Project a Series of LF Projects, 2025k). 2. *"Understand and incorporate discounted cost data into my reporting and analysis as appropriate in the rate optimization strategy"* (FinOps Foundation Project a Series of LF Projects, 2025k).	1. *"Ability to measure the overall effective savings rate of Cloud Rate Optimization efforts for both technology-based and monetary-based commitment discounts"* (FinOps Foundation Project a Series of LF Projects, 2025k). 2. *"Purchasing of commitment discounts are viewed as investments by stakeholder teams like Execs / Productment / Finance; the investment is the cost of the commitment over the entire period, and the return is the savings provided"* (FinOps Foundation Project a Series of LF Projects, 2025k).
Architecting for Cloud	Optimize	1. *"Maintain an awareness of cloud services, deployment patterns, development models, across my organization and generally"* (FinOps Foundation Project a Series of LF Projects, 2025k). 2. *"Follow and evaluate my workloads according to published or prescribed architectures"* (FinOps Foundation Project a Series of LF Projects, 2025a).	1. *"Measurement of cost efficiency through infrastructure savings, migration and support costs".* 2. *"Operational resiliency improvement in service quality and security risk posture"* (FinOps Foundation Project a Series of LF Projects, 2025a). 3. *"Enable a culture of rapid experimentation to drive innovation and cloud transformation"* (FinOps Foundation Project a Series of LF Projects, 2025a) (through Infrastructure as Code (Morris, 2020) for the architectural deployment).

Table 7.5: FinOps Capabilities, Activities and Phases mapping covered by the End-to-end Model in the FinOps Domain of Optimize Usage and Cost (FinOps Foundation Project a Series of LF Projects, 2025f)

.

7.3.1.3 FinOps for Containers Mapping

Containers are the smallest computational unit in the Cloud Native model (Amazon Web Services, 2024f) of a MACH Architecture (MACH Architecture, 2021b). In this context, understanding how container costs are structured and allocated is essential for mapping the model to the FinOps framework (FinOps Foundation Project a Series of LF Projects, 2025c). Therefore, a FinOps approach applied by Software Architects to MACH architectures should consider how to calculate container costs in alignment with the FinOps Principle that *"everyone takes ownership for their technology usage"* (FinOps Foundation Project a Series of LF Projects, 2025j), ensuring cost transparency and accountability across all teams. Here are the main cost components and strategies for allocating them:

Container Cost Analysis Components

Cost Type	Description and Allocation
Cluster Asset Costs	Total infrastructure cost for the container environment (compute, storage, network, etc.). Usually grouped by cluster or namespace and divided among workloads, overhead, and idle resources.
Workload Costs	Costs for running application containers or pods. Allocated based on the resources each workload uses, often tracked via labels or namespaces.
Overhead Costs	Costs for running the container platform itself (e.g., control plane, common services). These are shared among all users.
Idle Costs	Costs for unused, provisioned resources. These can be at the cluster, node, or pod level and represent inefficiencies.

Table 7.6: Key container cost components and allocation strategies in FinOps.

Key requirements for effective cost allocation: Accurate and timely data on what is running and its cost. Mechanisms to identify applications (labels, namespaces). Clear strategies for dividing overhead and idle costs. Ability to reconcile container costs with other application costs.

In the following tables, an example is shown based on the scope of the current cost-efficiency calculator:

EC2 worker node Replica Sizing — Variable Definitions

Symbol	Name	Description
N	Minimum replicas	Recommended minimum number of EC2 instances
R	Base RPS	Requests per second (Phase 1 input)
S_{req}	Request size (bytes)	Effective proto request size including gRPC/HTTP2 framing
S_{res}	Response size (bytes)	Effective proto response size including gRPC/HTTP2 framing
C_{req}	Throughput capacity (req/s)	Max requests/s per replica; defaults to $\lfloor C_{\text{out}}/S_{\text{res}} \rfloor$; can be specify by the software architect
C_{in}	Ingress bandwidth (bytes/s)	Network ingress capacity of the EC2 instance type
C_{out}	Egress bandwidth (bytes/s)	Network egress capacity of the EC2 instance type

Table 7.7: EC2 worker node replica sizing - variable definitions.

EC2 worker node Replica Sizing Formula

$$N = \max\left(\left\lceil \frac{R}{C_{\text{req}}} \right\rceil, \quad \left\lceil \frac{R \times S_{\text{req}}}{C_{\text{in}}} \right\rceil, \quad \left\lceil \frac{R \times S_{\text{res}}}{C_{\text{out}}} \right\rceil\right) \tag{7.1}$$

Dimension	Interpretation
$\lceil R/C_{\text{req}} \rceil$	**Throughput dimension.** Number of EC2 worker node replicas so that no single instance exceeds its request-processing capacity.
$\lceil (R \times S_{\text{req}})/C_{\text{in}} \rceil$	**Network ingress dimension.** Number of replicas so that aggregate inbound traffic does not saturate instance bandwidth.
$\lceil (R \times S_{\text{res}})/C_{\text{out}} \rceil$	**Network egress dimension.** Number of replicas so that aggregate outbound traffic does not saturate instance bandwidth. Typically the **bottleneck** for gRPC microservices because $S_{\text{res}} \gg S_{\text{req}}$.

Table 7.8: EC2 worker nodes replica sizing formula.

Monthly cost once N is known:

$$\text{Monthly EC2 cost} = p_{\text{hr}} \times 730 hours per month \times N$$

where p_{hr} is the on-demand hourly price from the AWS Pricing API.

References: OpenCost Specification (Ray, 2023); AWS EC2 Network Performance (Amazon Web Services, 2026b); VPA (Kubernetes, 2025); HPA (Kubernetes, 2026).

EC2 Worker Nodes Network Bandwidth References

Instance family / Ref	Network Specification	C_{out} (bytes/s)
`t3.medium` (Amazon Web Services, 2026c)	Up to 5 Gbps (Burst)	625,000,000
`m6i.large` (Amazon Web Services, 2026q)	12.5 Gbps (Up to) / Baseline Varies	1,562,500,000

Table 7.9: EC2 worker nodes network bandwidth baseline references.

EKS Cost Breakdown

Cost Component	Description and Allocation
EKS Control Plane	Fixed fee of $0.10/hour per cluster (~$73/month). Charged separately from worker nodes and not eligible for RI/SP discounts.
EC2 Worker Nodes	Compute cost for instances running workloads (e.g., t3.medium at $0.0416/hour in us-east-1). Billed as standard EC2 instances.
Reserved Instance Discount	Applies *only* to EC2 worker node costs (e.g., $30.37), not to the EKS control plane fee. Reduces the compute portion of the total EKS cost.
Total Monthly Cost	Sum of control plane (~$73) plus all EC2 worker nodes (~$30 per t3.medium). Example: 1 cluster + 1 node = ~$103/month before discounts.

Notes:

- The $73/month charge is *only* for the EKS control plane.
- EC2 worker nodes are billed *separately* as standard EC2 instances.
- Reserved Instances (RI) or Compute Savings Plans apply *only to the EC2 worker node costs*, not the EKS control plane fee.

References: AWS EKS Pricing: (Amazon Web Services, 2026d; Grande, 2025).

Table 7.10: EKS cost components showing separation between control plane and worker node charges.

MESSAGE TYPE	PROTO SIZE (BYTES)	EFFECTIVE SIZE (BYTES)	REQUESTS / MONTH	THROUGHPUT (GB/MONTH)	TRANSFER COST (USD/MONTH)
Request	258	258	3B	622.8089 GB	–
Response	7547	7547	3B	18,218.3683 GB	$ 1,599.76

Total Cost of Ownership (TCO) Breakdown

Cost Component	Category	Monthly Cost	% of TCO	ROI Impact
AWS Egress (Response Transfer) 1000 eff. RPS · AWS data-out tiers	Networking	$1599.76	91.4%	-92.0%
Application Load Balancer (ALB) $47.45/mo · /api/aws/alb-pricing	Cloud Infra	$47.45	2.7%	-2.7%
Containerized Cluster (EKS) $73.00/mo	Cloud Infra	$73.00	4.2%	-4.2%
EC2 Compute Replicas $30.37/mo	Cloud Infra	$30.37	1.7%	-1.7%
EC2 On-Demand (Standard RI 1-yr) (36.0% off) Discount applied to compute spend · /api/finops/ri-prices	FinOps Saving	-$10.93 saved	saving	+0.6%
Total Cost of Ownership (TCO)		**$1739.65/mo**	**100.0%**	**ROI Impact %**

Figure 7.7: TCO Breakdown report with EKS cost component showing separation between control plane and worker node (EC2 instance) charges with FinOps reserved instance 1 year applied.

Furthermore, when evaluating container environments, it is important to understand the different architectural patterns, as these affect both deployment complexity and costs. Here are some common container architectures, listed from simplest to most complex. **All-in-One single node:** All components (control plane, worker, storage) run on a single node. Best for development or small-scale, non-critical use. **Single Control Plane, Multiple Workers:** A single control plane manages several worker nodes. This setup is common and offers basic workload availability, but the control plane is still a single point of failure. **Single Control Plane with etcd, Multiple Workers:** Like the previous setup, but explicitly includes an `etcd` database on the control plane node. Improves role separation but still has a single point of failure. **Multiple Control Planes, Multiple Workers:** Introduces redundancy by having several control plane nodes, improving resilience and high availability. **Multiple Control Planes and etcd, Multiple Workers:** Both the control plane and `etcd` are distributed across multiple nodes. This is the most robust and suitable for mission-critical production environments.

The choice of architecture directly impacts both cluster asset costs and operational overhead, which are key considerations in the FinOps framework. Hence, it is expected for the FinOps Framework to take that ownership in a clear manner so that Software Architects and Engineers can avoid filling forms justifying resource costs and the allocation methodology to the Finance and Product teams. Hence, the end-to-end model helps them answer the following questions they are responsible for from a FinOps perspective. The questions can be found in this reference (The Linux Foundation, 2024): *"**What is the cost of running this entire environment?**"*: supported by the **cost calculation** and **forecasting** activities. *"**How efficient is that application or service? Where are there efficiencies to be found**"*: supported by the **FinOps optimization strategies** activity. *"**Why did we make certain design or architecture decisions that drive significant areas of costs?**"*: supported by **choosing architectural drivers** (architectural characteristics) and following the rest of the cycle per each of them. *"**What's the cost of all the***

share infrastructure? How should we divided that up, budget for it, forecast it into the future?": supported by the **cost calculation** and **forecasting activities** in each cycle.

7.3.2 AWS Well-Architected Framework Mapping

According to the **6 pillars** presented in the **AWS Well-Architected Framework** (Eliot & Valverde, 2018), the following are supported by the model:

AWS Well-Architected Model Mapping

AWS Pillar	Pillar Practice	Pillar Subpractices	How are those supported?	Design Principles
Security Pillar (Amazon Web Services, 2024d).	Data Protection (Amazon Web Services, 2026g).	**SEC09-BP01** Implement secure key and certificate management (Amazon Web Services, 2026n). **SEC09-BP02** Enforce encryption in transit (Amazon Web Services, 2026o).	The model covers the secure service-to-service communication via gRPC using TLS 1.3 or mTLS, which implies the use of **Amazon Certificate Manager** (Amazon Web Services, 2026e).	Protect data in transit and at rest (Amazon Web Services, 2026p).

Table 7.11: AWS Well-Architected Model Mapping - Security Pillar

.

AWS Well-Architected Model Mapping

AWS Pillar	Pillar Practice	Pillar Subpractices	How are those supported?	Design Principles
Reliability Pillar (Amazon Web Services, 2026l).	Failure Management (Amazon Web Services, 2023c).	**REL11-BP02** Fail over to healthy resources (Amazon Web Services, 2026k).	The model associates Elastic Load Balancing pattern as suggested in (Amazon Web Services, 2026k) through the Application Load Balancer service (Amazon Web Services, 2026s).	Scale horizontally to increase aggregate workload availability (Amazon Web Services, 2026h).

Table 7.12: AWS Well-Architected Model Mapping - Reliability Pillar.

AWS Well-Architected Model Mapping

AWS Pillar	Pillar Practice	Pillar Subpractices	How are those supported?	Design Principles
Cost Opt. Pillar (Amazon Web Services, 2025q).	Practice Cloud Financial Mgmt. (Amazon Web Services, 2025u).	**COST01-BP02** Establish a partnership between finance and technology (Amazon Web Services, 2025b). **COST01-BP03** Establish cloud budgets and forecasts (Amazon Web Services, 2025c). **COST01-BP04** Implement cost awareness in your organizational processes (Amazon Web Services, 2025d). **COST01-BP05** Report and notify on cost optimization (Amazon Web Services, 2025e). **COST01-BP09** Quantify business value from cost optimization (Amazon Web Services, 2025f).	The model outcomes are cost calculations and forecasting for the services chosen to build the MACH Architecture. Those represent part what the software architect needs to establish effective communication with the finance and technology teams, in the Design phase from the technological perspective.	Implement Cloud Financial Management (Amazon Web Services, 2025r).

Table 7.13: AWS Well-Architected Model Mapping - Cost Optimization Pillar.

AWS Well-Architected Model Mapping

AWS Pillar	Pillar Practice	Pillar Subpractices	How are those supported?	Design Principles
Cost Opt. Pillar (Amazon Web Services, 2025q).	Cost-effective resources (Amazon Web Services, 2025p).	**COST05-BP01** Identify organization requirements for cost (Amazon Web Services, 2025g). **COST05-BP05** Select components of this workload to optimize cost in line with organization priorities (Amazon Web Services, 2025h). **COST05-BP06** Perform cost analysis for different usage over time (Amazon Web Services, 2025i). **COST06-BP01** Perform cost modeling (Amazon Web Services, 2025j).	The model outcomes are cost calculations and forecasting for the services chosen to build the MACH Architecture, as well as options for cost optimization FinOps strategies over those services. In each cycle in which an architectural driver like reliability or resiliency has been chosen, the balanced with cost optimization is supported as well in each cost calculation and the rest of the outcomes that can be compared with previous cycles (other architectural decisions). The cost calculations are performed in the model based on billing and cost management tools, and pricing calculators.	**Adopt a consumption model**, and **Analyze and attribute expenditure** (Amazon Web Services, 2025r).

Table 7.14: AWS Well-Architected Model Mapping - Cost Optimization Pillar (Part 2).

AWS Well-Architected Model Mapping

AWS Pillar	Pillar Practice	Pillar Subpractices	How are those supported?	Design Principles
Cost Opt. Pillar (Amazon Web Services, 2025q).	Cost-effective resources (Amazon Web Services, 2025p).	**COST06-BP02** Select resource type, size, and number based on data (Amazon Web Services, 2025k). **COST07-BP04** Implement pricing models for all components of this workload (Amazon Web Services, 2025l). **COST07-BP05** Perform pricing model analysis at the management account level (Amazon Web Services, 2025m).	The model outcomes are cost calculations and forecasting for the services chosen to build the MACH Architecture, as well as options for cost optimization FinOps strategies over those services. In each cycle in which an architectural driver like reliability or resiliency has been chosen, the balanced with cost optimization is supported as well in each cost calculation and the rest of the outcomes that can be compared with previous cycles (other architectural decisions). The cost calculations are performed in the model based on billing and cost management tools, and pricing calculators.	**Adopt a consumption model**, and **Analyze and attribute expenditure** (Amazon Web Services, 2025r).

Table 7.15: AWS Well-Architected Model Mapping - Cost Optimization Pillar (Part 3).

AWS Well-Architected Model Mapping

AWS Pillar	Pillar Practice	Pillar Subpractices	How are those supported?	Design Principles
Cost Opt. Pillar (Amazon Web Services, 2025q).	Manage demand and supply resources (Amazon Web Services, 2025t).	**COST09-BP01** Perform an analysis on the workload demand (Amazon Web Services, 2025n). **COST09-BP02** Implement a buffer or throttle to manage demand (Amazon Web Services, 2025o).	The model considers Retry as one of the architectural tactics and patterns for resilient communication for gRPC-based MACH architectures.	Implement Cloud Financial Management (Amazon Web Services, 2025r).

Table 7.16: AWS Well-Architected Model Mapping - Cost Optimization Pillar (Part 4).

7.4 Sequence Diagrams of the main activities of the model operationalization process

In this section, I present the sequence diagrams of the main activities of the model operationalization process and the networking cost calculation for ProtocolBuffer files of gRPC in detail. The diagramming tool that I used was Visual Paradigm (2026) for non commercial use. For the notation, I followed the suggestions presented by Martin (2003).

Note: Only the endpoints invoked by the SoftwareArchitect actor that are related to HTTP POST methods were specified; the rest of the API calls are HTTP GET methods. The notation specification for all the sequence diagrams was the following:

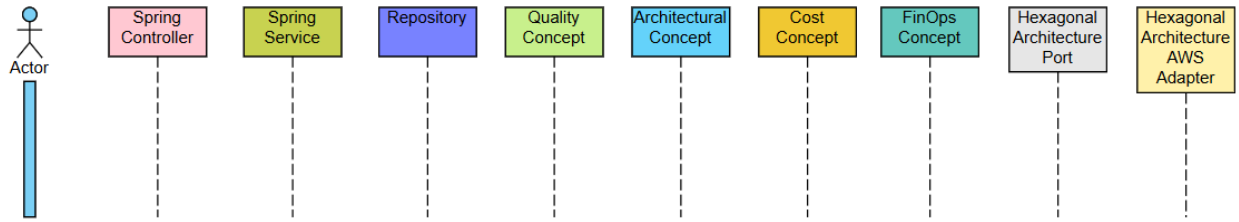


Figure 7.8: Sequence Diagrams Notation Specification.

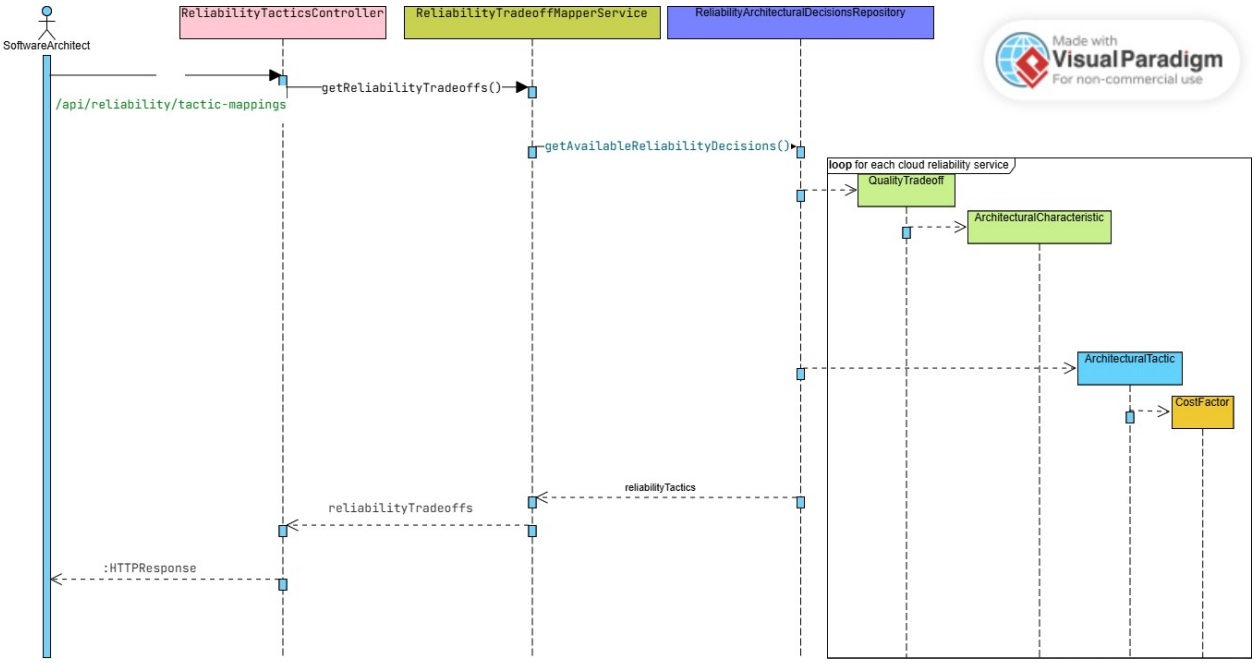


Figure 7.9: Activities 1 and 2 - Choose Quality Attribute and Tactics to promote them - Reliability tactics example - Sequence Diagram.

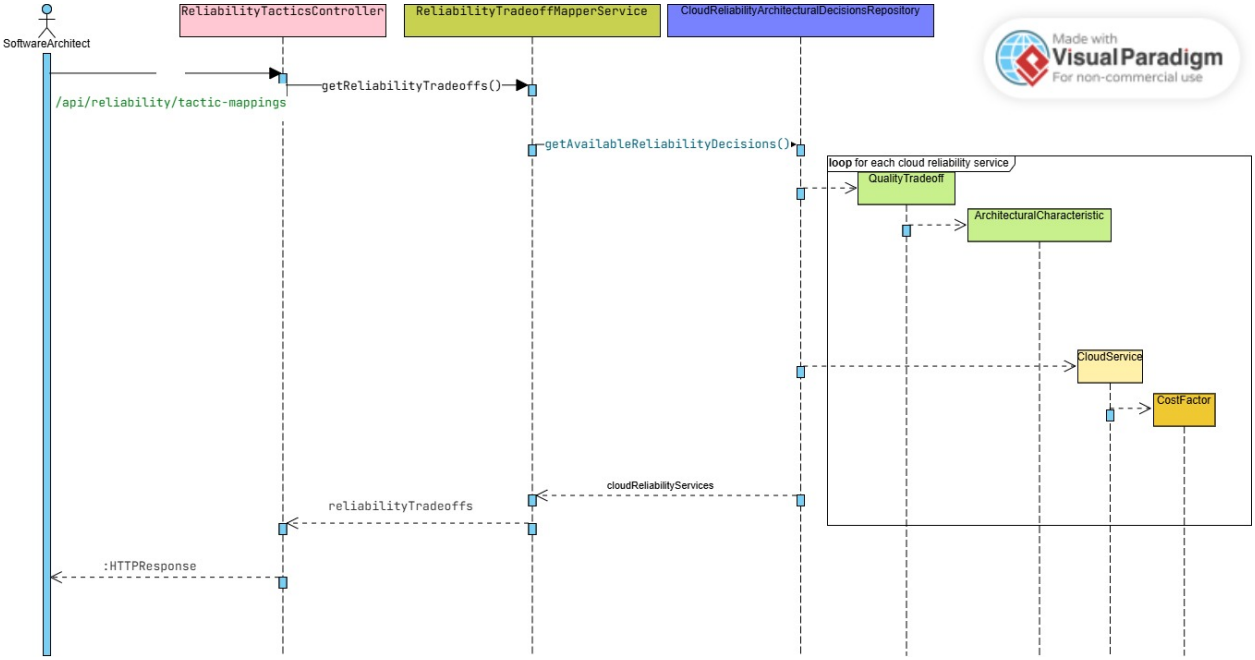


Figure 7.10: Activity 3 - Map Cloud Tactics - Cloud Reliability tactics example - Sequence Diagram.

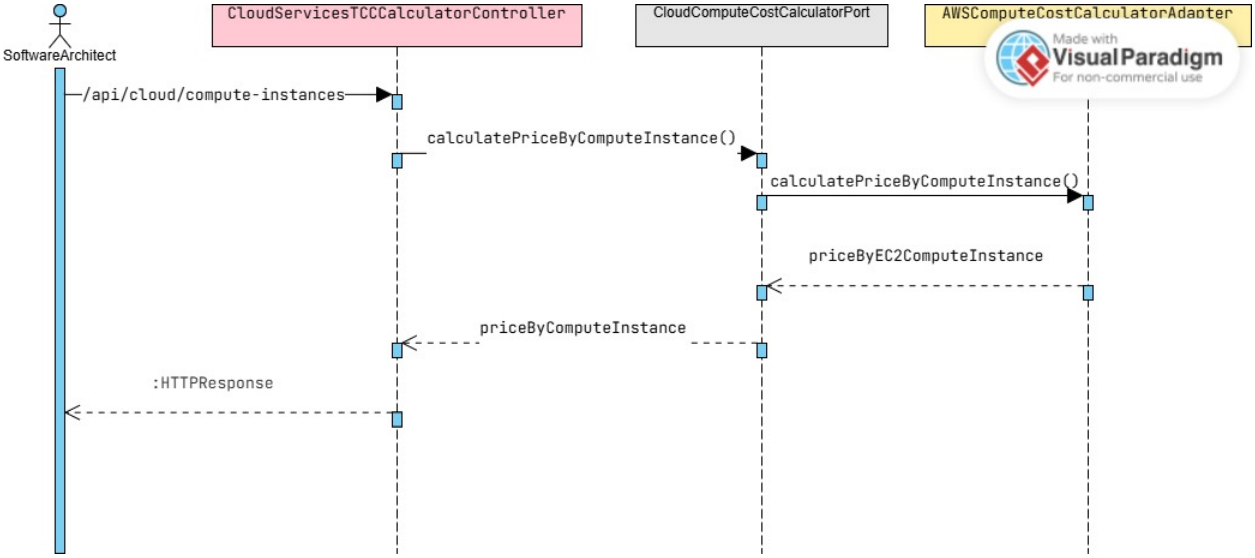


Figure 7.11: Cost calculation and optimization activities 4, 5, 6, and 7 - TCC Cost calculation - Sequence Diagram - Price by compute instance - AWS EC2 example.

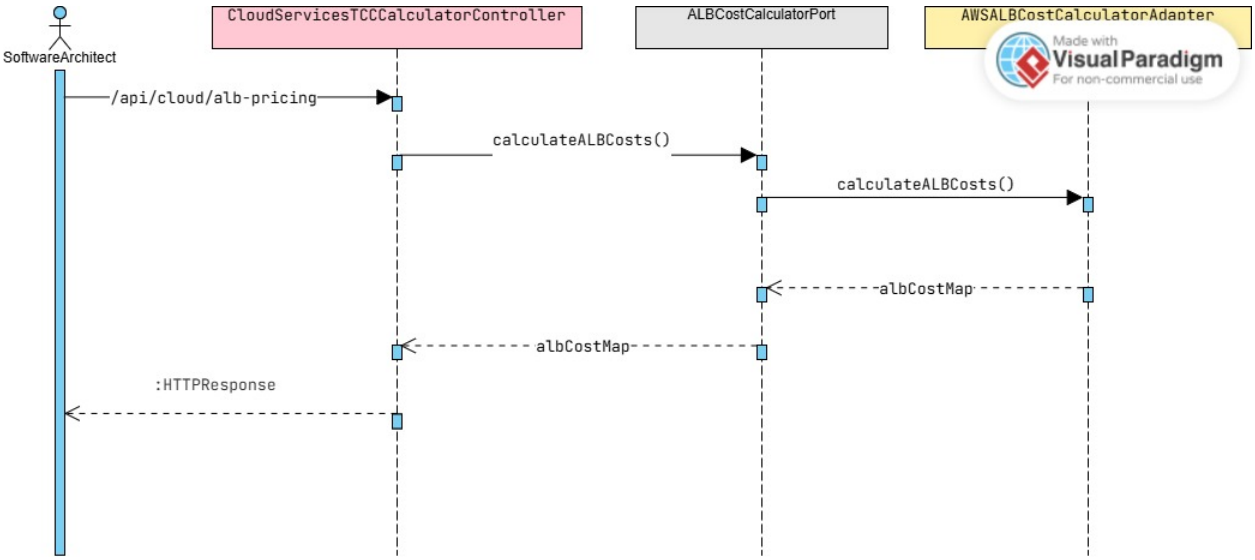


Figure 7.12: Cost calculation and optimization activities 4, 5, 6, and 7 - TCC Cost calculation - Sequence Diagram - Load Balancing pricing - AWS ELB/ALB example.

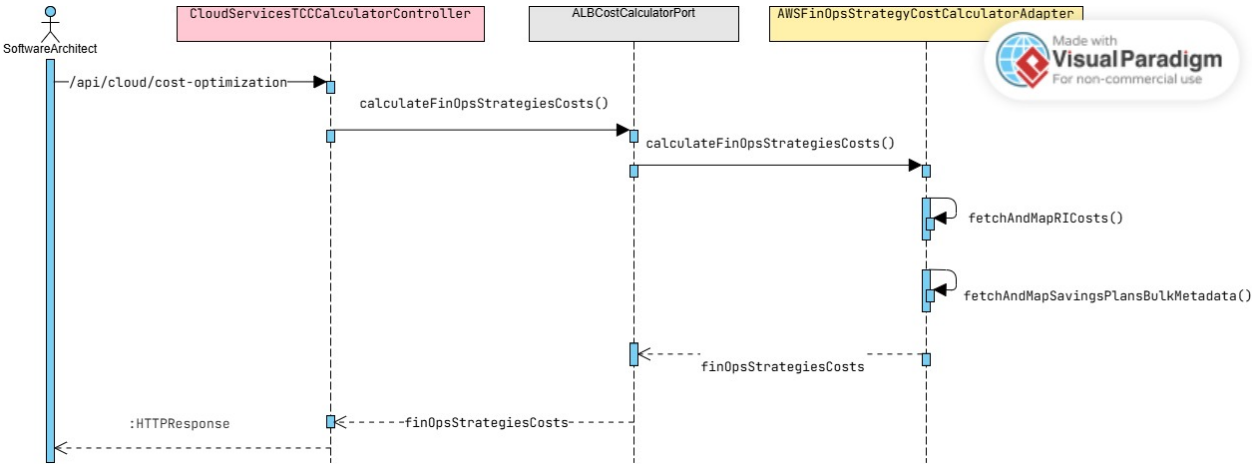


Figure 7.13: Cost calculation and optimization activities 4, 5, 6, and 7 - TCC Cost calculation - Sequence Diagram - FinOps strategies costs - AWS FinOps Reserved Instances and Compute Savings Plans strategies example.

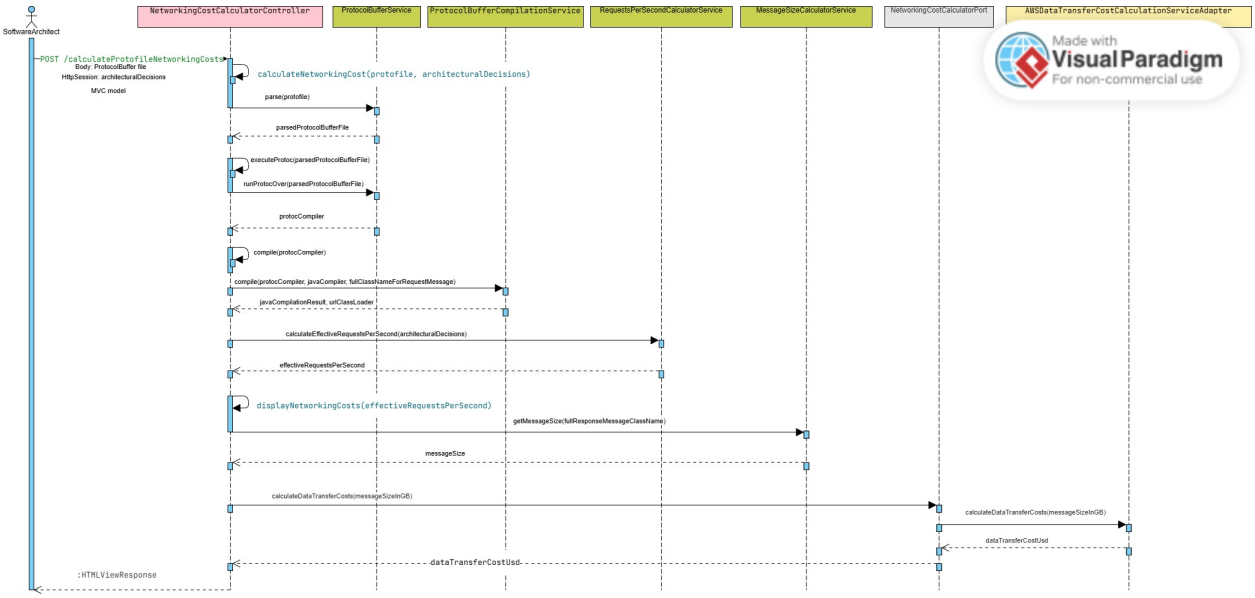


Figure 7.14: Cost calculation and optimization - Networking Cost Calculation for Protocol Buffer files - Sequence Diagram.

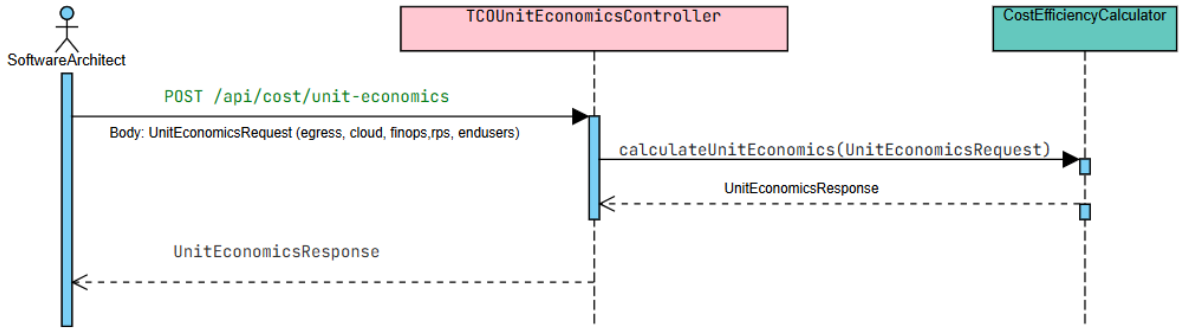


Figure 7.15: Cost calculation and optimization - TCO and Unit economics calculation - Sequence Diagram.

7.5 SBE specifications for resiliency and security architectural characteristics

Phase 1 – Service Identity: User Story: Base RPS Cost Projections

As a Software Architect
So that I can understand the financial baseline of my service workload
I want to enter a base RPS and see how tactic-driven load increases affect egress costs

Scenario Outline: Base RPS drives egress cost delta when a tactic increases effective load
Given a service with ***<base_rps>*** requests per second
When the architect selects a tactic that increases effective RPS by ***<rps_delta>***
Then the live cost delta panel shows an egress cost increase of approximately ***<expected_delta_usd>*** per month

Phase 1 – Workload Cost Analysis

base_rps	rps_delta	expected_delta_usd
1000	+50 RPS (5% Retry Profile)	$13.02 / mo
5000	+250 RPS (5% Retry Profile)	$65.17 / mo

Table 7.17: Phase 1 – Service Identity: Workload Scenarios.

Phase 2 – Service Contracts: User Story: Proto Contract Auto-Loading

As a Software Architect
So that the backend can derive exact serialized message sizes
I want to select my gRPC communication pattern and have the corresponding .proto IDL verified as pre-loaded automatically

Scenario: Selecting a standard BUC loads the correct proto contract
Given four canonical Business Use Cases (BUC) are available, each mapping to a gRPC streaming pattern
When the architect clicks a BUC card ”BUC1 - Retrieve Order by ID”
Then the tool automatically loads ”unaryRPCPattern.proto” into application state
And the proto filename is displayed in the UI as ” unaryRPCPattern.proto (pre-loaded)”
And the architect may override it by uploading a custom .proto file

Scenario Outline: BUC-to-proto mapping for the four gRPC patterns
Given the architect selects ***<buc_id>***
When the BUC card is activated
Then ***<proto_file>*** is loaded and ***<rpc_type>*** is shown as the pattern badge

Phase 2 – gRPC Schema Contracts

buc_id	use_case	proto_file	rpc_type
BUC1	Retrieve Order by ID	unaryRPCPattern.proto	Unary RPC
BUC2	Search & Filter Orders	serverStreamingRPCPattern .proto	Server Streaming
BUC3	Bulk Update Orders	clientStreamingRPCPattern .proto	Client Streaming
BUC4	Continuous Shipment Consolidation	biDirectionalStreamingRPC Pattern.proto	Bi-Directional

Table 7.18: Phase 2 – BUC to Protocol Buffer IDL files mappings.

unaryRPCPattern.proto

```
23
24  message OrderItem {
25    string product_id = 1;
26    int32 quantity = 2;
27    Money unit_price = 3;
28  }
29
30  message Address {
31    string street = 1;
32    string city = 2;
33    string state = 3;
34    string zip_code = 4;
35    string country = 5;
36  }
37
38  message Order {
39    string order_id = 1;
40    string user_id = 2;
41    repeated OrderItem items = 3;
42    Money total_amount = 4;
43    google.protobuf.Timestamp order_date = 5;
44    OrderStatus status = 6;
45    Address shipping_address = 7;
46  }
47
48  message GetOrderRequest {
49    string order_id = 1;
50  }
51
52  service UnaryOrderService {
53    rpc GetOrderDetails (GetOrderRequest) returns (Order);
54  }
```

Figure 7.16: BUC1: gRPC UnaryPattern: Retrieve orders by using an order ID from client to server - protocol buffer IDL.

serverStreamingRPCPattern.proto

```
10      enum OrderStatus {
15        ORDER_STATUS_DELIVERED = 4;
16        ORDER_STATUS_CANCELLED = 5;
17      }
18
19      message Money {
20        int64 units = 1;
21        int32 nanos = 2;
22      }
23
24      message SearchOrdersRequest {
25        string query = 1;
26        OrderStatus status_filter = 2;
27        google.protobuf.Timestamp start_date = 3;
28        google.protobuf.Timestamp end_date = 4;
29        Money min_amount = 5;
30      }
31
32      message OrderSummary {
33        string order_id = 1;
34        string user_id = 2;
35        Money total_amount = 3;
36        google.protobuf.Timestamp order_date = 4;
37        OrderStatus status = 5;
38        string tracking_number = 6;
39      }
40
41      service OrderSearchService {
42        rpc SearchOrders (SearchOrdersRequest) returns (stream OrderSummary);
43      }
```

Figure 7.17: BUC2: gRPC Server Streaming: The business needs to retrieve all possible orders that match a search criterion (term or filter) - protocol buffer IDL.

clientStreamingRPCPattern.proto

```
43    message OrderUpdateRequest {
46      OrderStatus new_status = 3;
47      string status_reason = 4;
48      string note = 5;
49      ShippingUpdate shipping_update = 6;
50      double new_amount = 7;
51      string amount_adjustment_reason = 8;
52      string updated_by = 9;
53      google.protobuf.Timestamp request_time = 10;
54    }
55
56    message UpdateResult {
57      string order_id = 1;
58      bool success = 2;
59      string error_message = 3;
60      OrderStatus previous_status = 4;
61      OrderStatus current_status = 5;
62      google.protobuf.Timestamp updated_at = 6;
63    }
64
65    message BatchUpdateResponse {
66      int32 total_received = 1;
67      int32 successful_updates = 2;
68      int32 failed_updates = 3;
69      repeated UpdateResult results = 4;
70      google.protobuf.Timestamp processing_started = 5;
71      google.protobuf.Timestamp processing_completed = 6;
72      double processing_duration_seconds = 7;
73    }
74
75    service OrderBatchUpdateService {
76      rpc BatchUpdateOrders (stream OrderUpdateRequest) returns (BatchUpdateResponse);
77    }
```

Figure 7.18: BUC3: gRPC Client Streaming pattern: Update a set of orders - protocol buffer IDL.

biDirectionalStreamingRPCPattern.proto

```
20
21  message Address {
22    string street = 1;
23    string city = 2;
24    string state = 3;
25    string zip_code = 4;
26    string country = 5;
27  }
28
29  message OrderProcessingRequest {
30    string order_id = 1;
31    string user_id = 2;
32    repeated OrderItem items = 3;
33    Address shipping_address = 4;
34    google.protobuf.Timestamp requested_delivery_date = 5;
35  }
36
37  message CombinedShipment {
38    string shipment_id = 1;
39    repeated string order_ids = 2;
40    google.protobuf.Timestamp delivery_date = 3;
41    Address shipping_address = 4;
42    Money total_shipping_cost = 5;
43    int32 total_items = 6;
44    google.protobuf.Timestamp created_at = 7;
45  }
46
47  service BidirectionalOrderProcessingService {
48    rpc ProcessOrdersForCombinedShipments (stream OrderProcessingRequest) returns (stream CombinedShipment);
49  }
```

Figure 7.19: BUC4: gRPC Bi-Directional Streaming pattern: Send a continuous set of orders and process them into combined shipments based on the delivery date - protocol buffer IDL.

Phase 3 – Architectural Security: User Story: Security Tactic Configurations

As a Software Architect
So that I can account for the byte overhead of encryption and authentication in my TCO
I want to enable security tactics and observe their impact on message sizes and egress costs

Scenario Outline: Security tactics OWASP mapping
Given the architect enables ***<security_tactic>***
Then it addresses owasp category ***<owasp_categories>*** and adds ***<overhead>*** to each message

Phase 3 – Security Compliance Overhead

security_tactic	owasp_categories	overhead
TLS (One-way)	Cryptographic Failures	30 B
mTLS (Mutual TLS)	Broken Access Control	5 TLS messages
OAuth 2.0 + JWT	Identification and Authentication Failures	650 B
Basic Authentication	Identification and Authentication Failures	0

Table 7.19: Phase 3 – Security Tactic Configurations.

Phase 3 – Security Tactic Configurations

security tactic	overhead calculation basis	calculated overhead
TLS (One-way)	RFC 8446 header (5B) + AEAD Nonce (8B) + AEAD Tag (16B) + 1B Content-type (McGrew, 2008; Rescorla, 2026; Rescorla & Rescorla, 2018)	30 B
mTLS (Mutual TLS)	Base TLS handshake + CertRequest, Client Cert, and CertificateVerify messages (Rescorla, 2026; Rescorla & Rescorla, 2018).	5 TLS messages
OAuth 2.0 + JWT	Header (≈ 70B) + Payload (≈ 480B) + ES256 Signature (≈ 100B) (Jones, Bradley, & Sakimura, 2015; Jones, Bradley, Sakimura, Jones, et al., 2015).	650 B
Basic Authentication	Standard string encoding (base64).	0

Table 7.20: Phase 3 – Security Tactic Configurations - RFC references.

Phase 4 - Unit Economics report

As a Software Architect
So that I can evaluate the cost-efficiency of the microservice I modelled
I want to calculate the unit economics

Scenario Outline: Validate Core Unit Economics Calculations (ARPU $15.00)
Given a unit economics request with egress cost ***<egress>***, cloud cost ***<cloud>***, RPS ***<rps>***, consumers ***<consumers>***, and ARPU ***<arpu>***
When the unit economics are calculated
Then the response returns status 200
And the calculated gross parameters match monthly TCO ***<expected_monthly_tco>*** and annual TCO ***<expected_annual_tco>*** with requests ***<expected_requests>***
And the unit cost parameters match cost per request ***<expected_cost_per_req>***, cost per user month ***<expected_cost_user_month>***, and cost per user day ***<expected_cost_user_day>***
And the ROI metrics match total monthly revenue ***<expected_monthly_rev>***, monthly profit ***<expected_monthly_profit>***, monthly ROI pct ***<expected_roi_pct>***, break-even users ***<expected_breakeven>***, and net margin per user ***<expected_net_margin>***

Phase 4 - Unit Economics examples

egr	cld	rps	con	arpu	tco	rq_m	c_rq	c_usr	rv_m	roi	b_ev	net
1599.7	66.89	1000	350	7.0	1666.6	2.59B	1.0e-6	4.76	2.45k	47.0%	239	2.23
185.23	5257.9	1050	350	15.0	5443.2	2.72B	2.00e-6	15.55	5.25k	-3.55%	363	-0.55

Table 7.21: Phase 4 - Unit economics report examples.

7.5.1 Total Cost of Ownership and Unit Economics formulas

Finally, the core unit economics, and TCO calculation formulas are detailed in the following table.

On one hand, the **Cost per User** represents the operational infrastructure cost allocated per individual customer. It's not properly **Customer Acquisition Cost (CAC)** because it is defined as *"the average cost of attracting one client"* (Khanin, 2024), but in terms of the purpose of this model the cost per user might contribute to it from an **infrastructure costs** perspective.

On the other hand, the **Net Margin** indicates the profit or loss generated per user after infrastructure overhead.

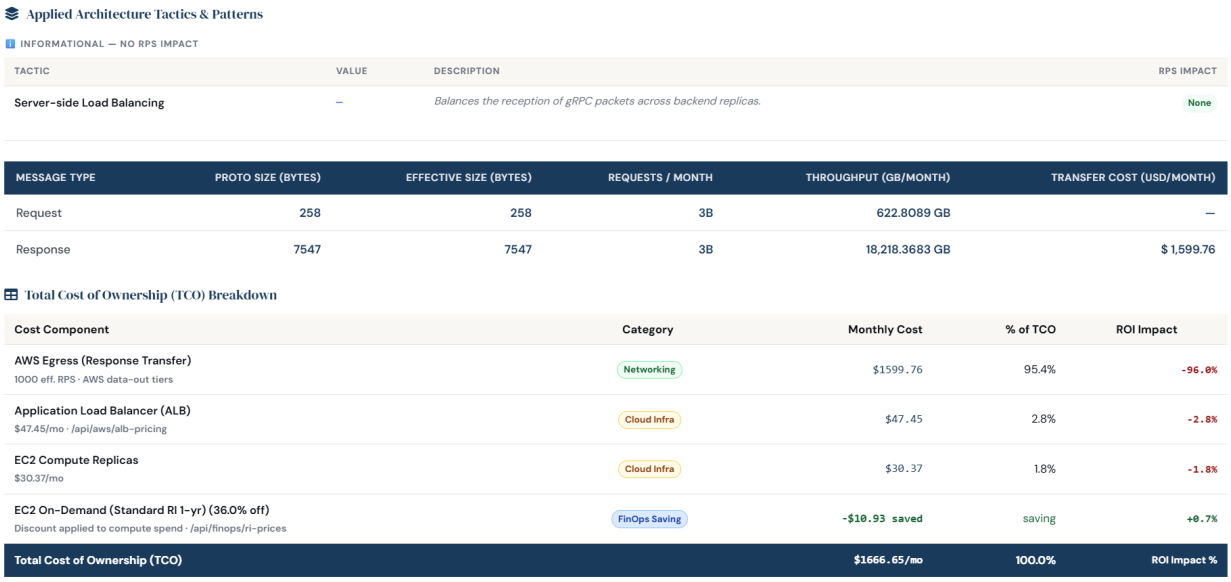
Unit Economics and Cost per User Metrics

Metric	Formula / Description
Monthly TCO (TCO_{mo})	Egress (Networking costs) + Cloud Infrastructure
Monthly Requests (Req_{mo})	$RPS \times 86,400 seconds per day \times 30 days per month$
Cost per Request (C_{req})	$TCO_{\text{mo}}/Req_{\text{mo}}$
Cost per User	TCO_{mo}/End users
Net Profit Per User (Net Margin (NM) numerator)	$ARPU$ − Cost per User
Monthly Revenue (Rev_{mo})	End users × ARPU
Net Monthly Profit (P_{mo})	$Rev_{\text{mo}} - TCO_{\text{mo}}$
Monthly ROI % (ROI)	$(P_{\text{mo}}/TCO_{\text{mo}}) \times 100$
Break-Even Users (BE)	$\lceil TCO_{\text{mo}}/\text{ARPU} \rceil$

References: **Net Profit or Net Income**: (Kenton, 2026). **ROI**: (Fernando, 2026). **Break-Even Users**: (U.S. Small Business Administration, 2024) considering variable costs as zero.

Table 7.22: Unit economics and cost per user metrics.

Following the example presented in the SBE definitions above for the 7 activities the software architect should perform, the following are the TCO Breakdown and the Unit economics views:



Applied Architecture Tactics & Patterns

INFORMATIONAL — NO RPS IMPACT

TACTIC	VALUE	DESCRIPTION	RPS IMPACT
Server-side Load Balancing	–	*Balances the reception of gRPC packets across backend replicas.*	None

MESSAGE TYPE	PROTO SIZE (BYTES)	EFFECTIVE SIZE (BYTES)	REQUESTS / MONTH	THROUGHPUT (GB/MONTH)	TRANSFER COST (USD/MONTH)
Request	258	258	3B	622.8089 GB	–
Response	7547	7547	3B	18,218.3683 GB	$ 1,599.76

Total Cost of Ownership (TCO) Breakdown

Cost Component	Category	Monthly Cost	% of TCO	ROI Impact
AWS Egress (Response Transfer) 1000 eff. RPS · AWS data-out tiers	Networking	$1599.76	95.4%	-96.0%
Application Load Balancer (ALB) $47.45/mo · /api/aws/alb-pricing	Cloud Infra	$47.45	2.8%	-2.8%
EC2 Compute Replicas $30.37/mo	Cloud Infra	$30.37	1.8%	-1.8%
EC2 On-Demand (Standard RI 1-yr) (36.0% off) Discount applied to compute spend · /api/finops/ri-prices	FinOps Saving	-$10.93 saved	saving	+0.7%
Total Cost of Ownership (TCO)		**$1666.65/mo**	**100.0%**	**ROI Impact %**

Figure 7.20: TCO Breakdown report for server side load balancing and Reserved Instances mapped to EC2.

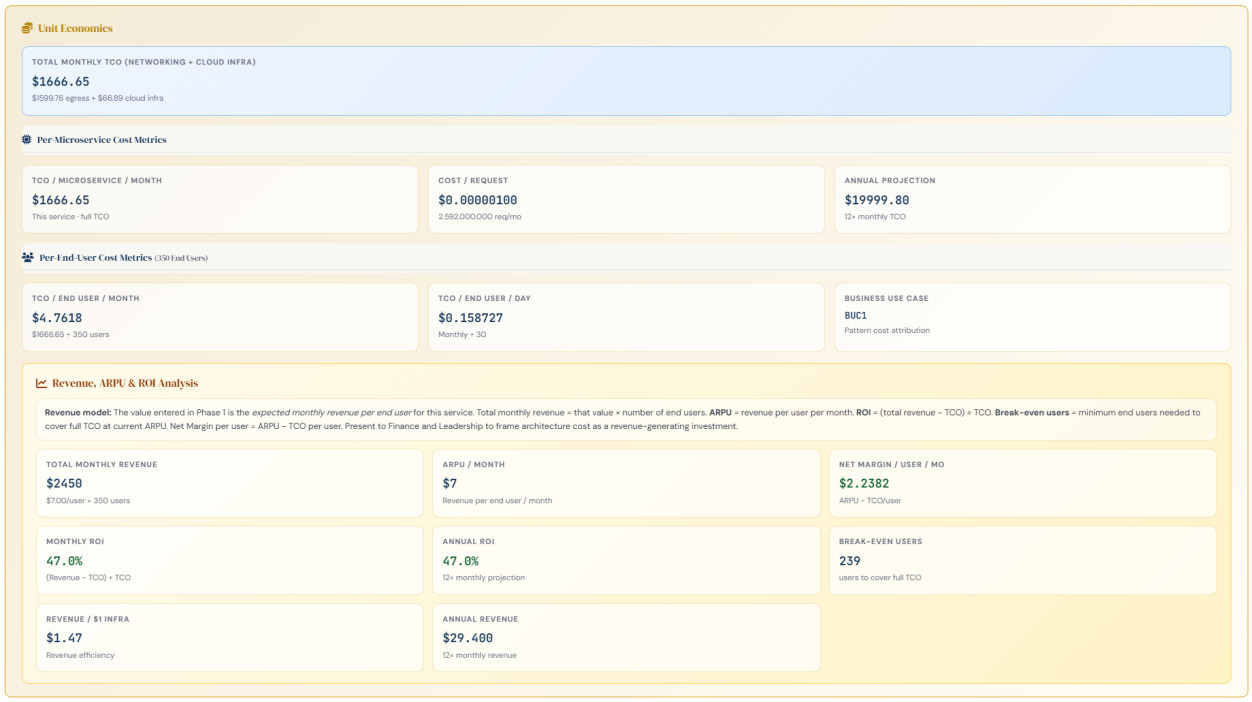


Figure 7.21: Unit economics report for server side load balancing and Reserved Instances mapped to EC2.

7.6 MACH Architecture Portfolio TCO report with Unit economics

As an extension of the scope of this thesis, a MACH Architecture portfolio TCO and unit economics breakdown is offered by modelling multiple microservices.

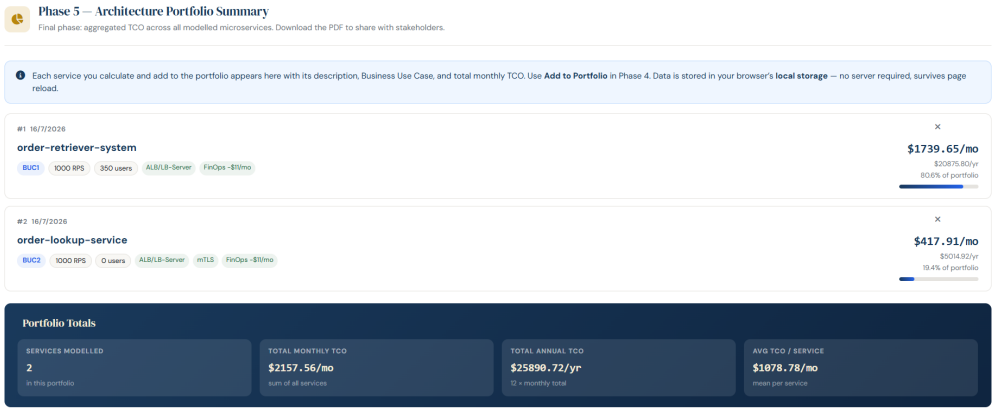


Figure 7.22: Portfolio TCO Summary report.

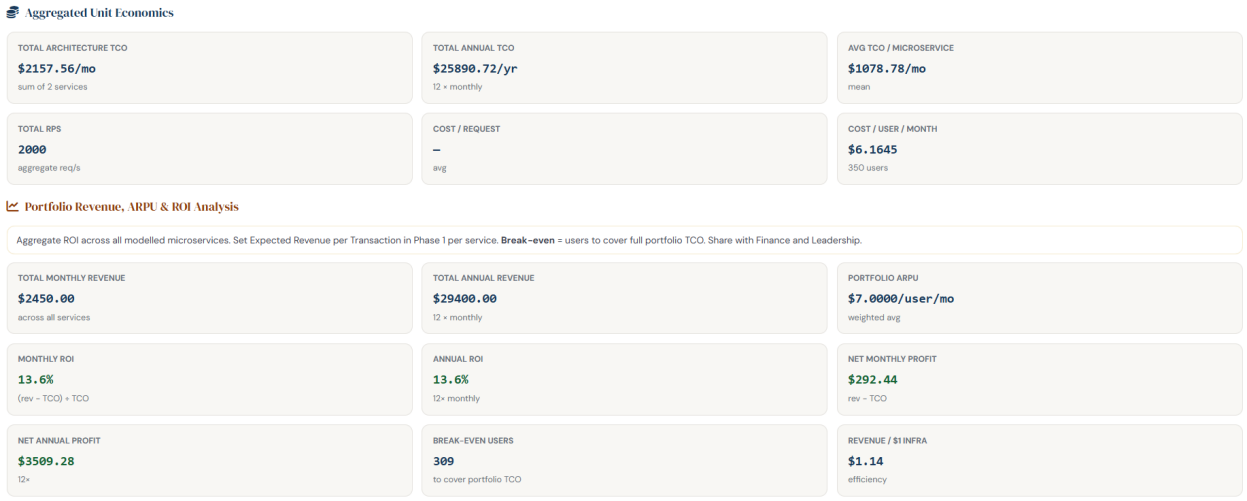


Figure 7.23: Portfolio Unit economics summary report.

7.7 GitHub Repository

The following is the private **GitHub Repository including the software architect's tool to evaluate cost efficiency and the 4 protocol buffer files representing the four Business Use Cases**: (Chávez González, 2025).

Some screenshots of the implementation are presented in the following figures for technically supporting this implementation.

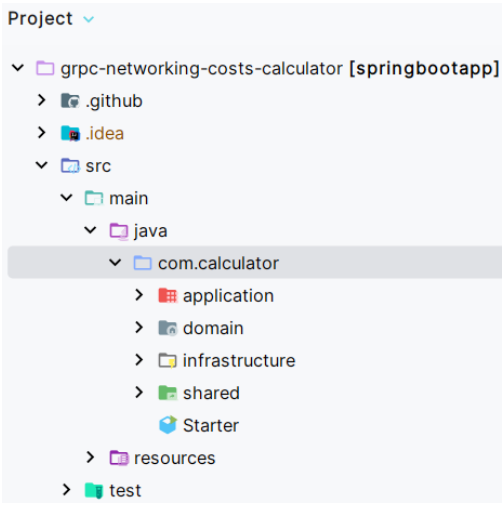


Figure 7.24: Software Architecture - Architectural Layers.

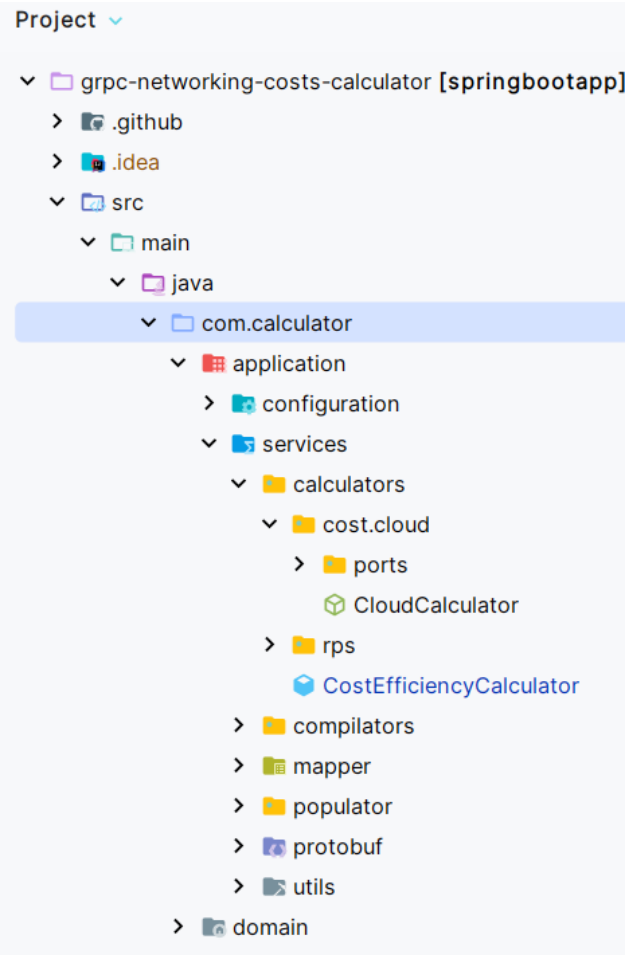


Figure 7.25: Software Architecture - Architectural Layers - Application Layer.

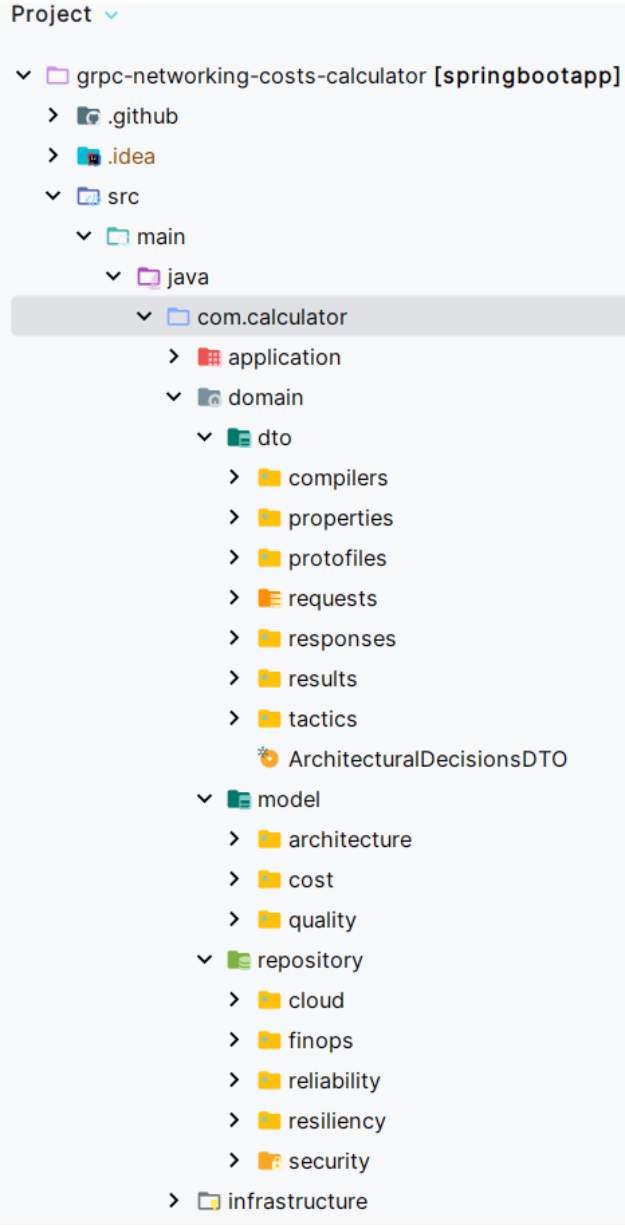


Figure 7.26: Software Architecture - Architectural Layers - Domain Layer.

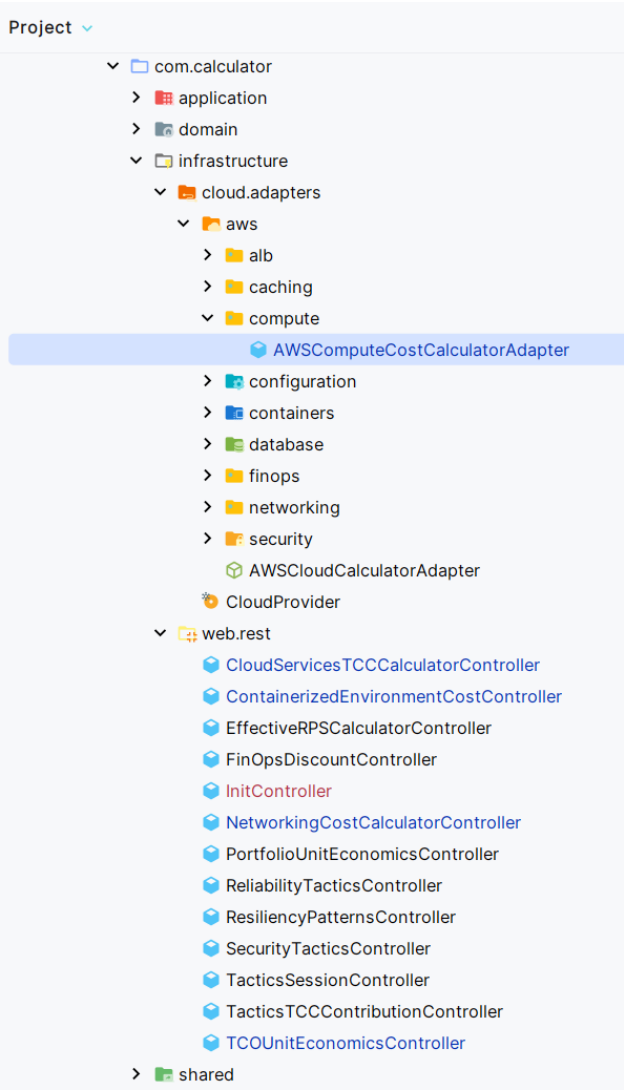


Figure 7.27: Software Architecture - Architectural Layers - Infrastructure Layer.

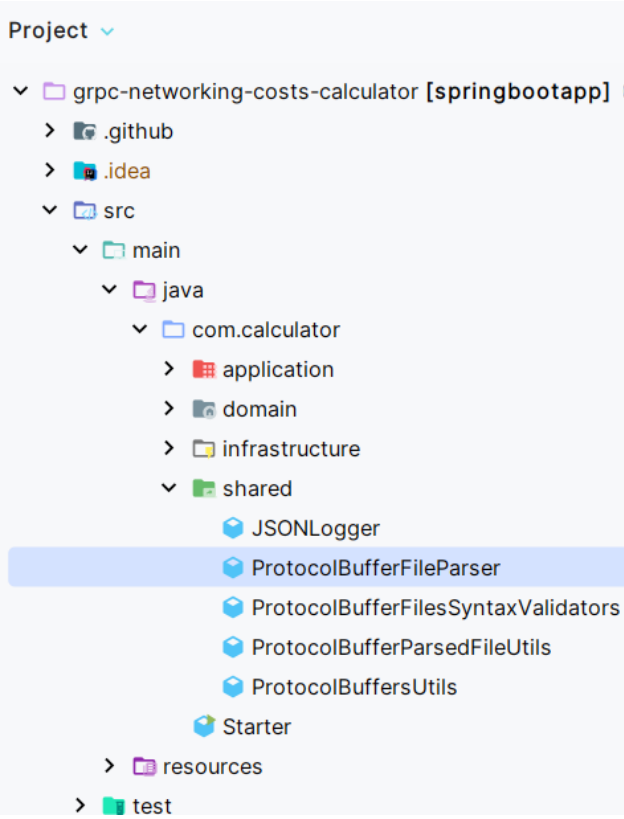


Figure 7.28: Software Architecture - Architecture Layers - Shared Layer.

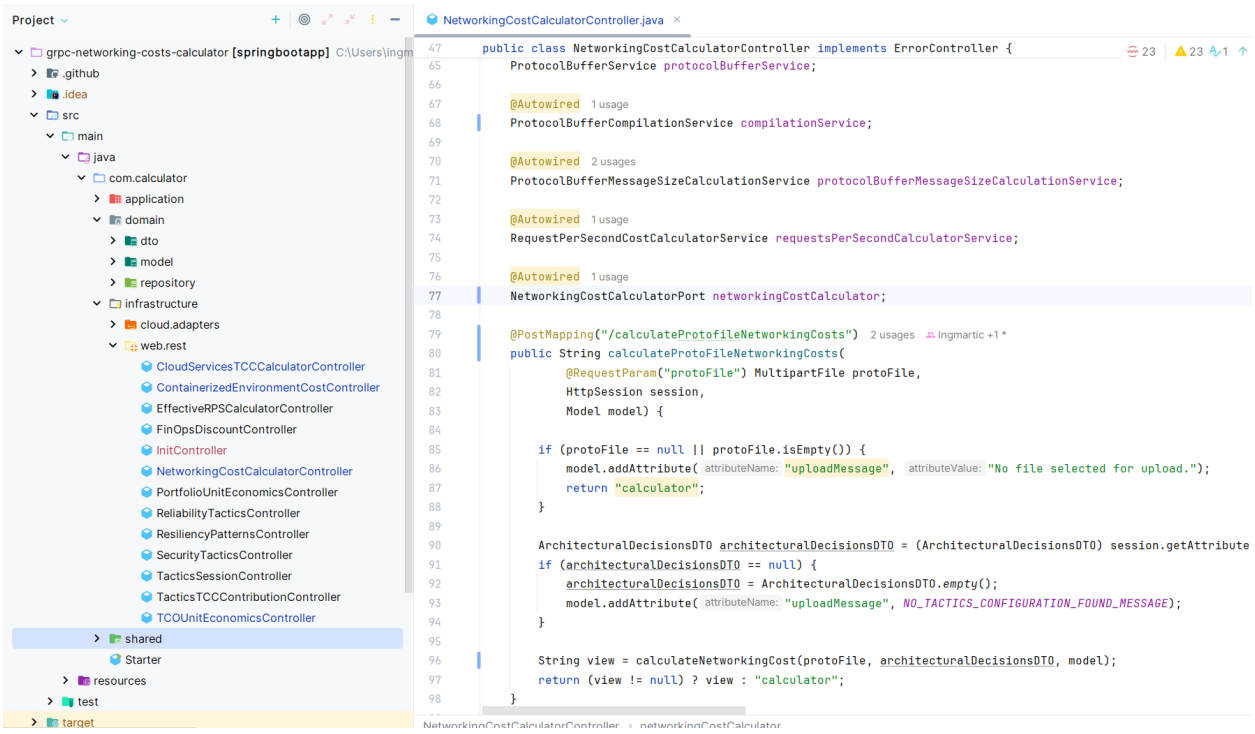


Figure 7.29: MVC and REST controllers - NetworkingCostCalculatorController.

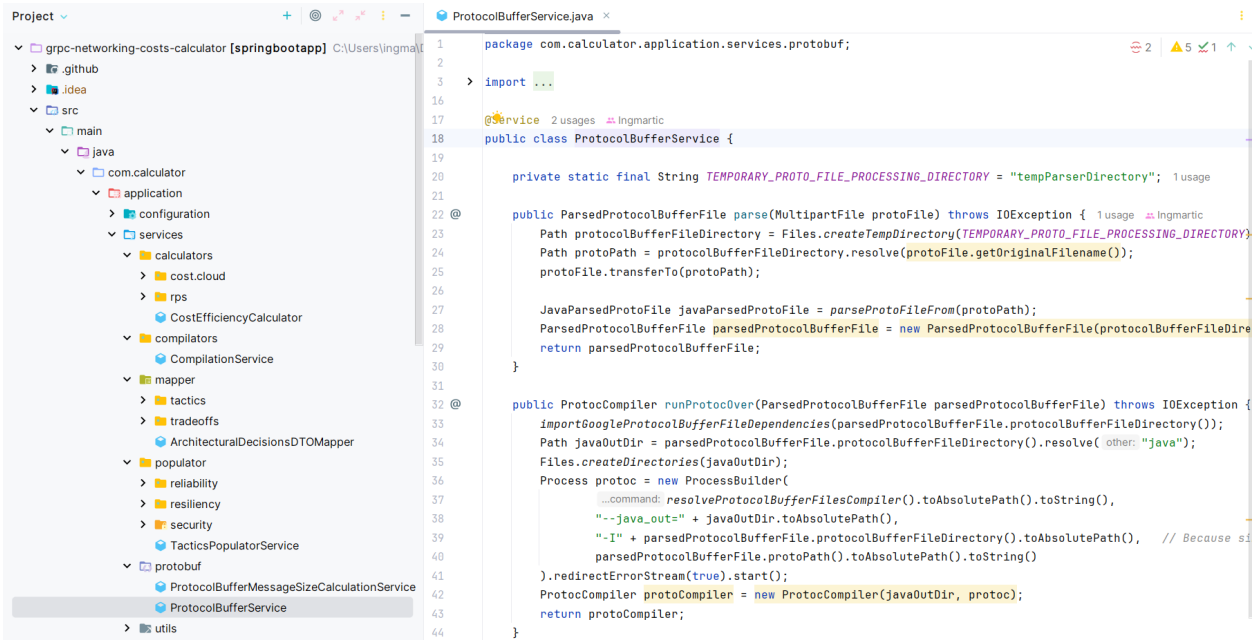


Figure 7.30: Protocol Buffer service - parser and loader.

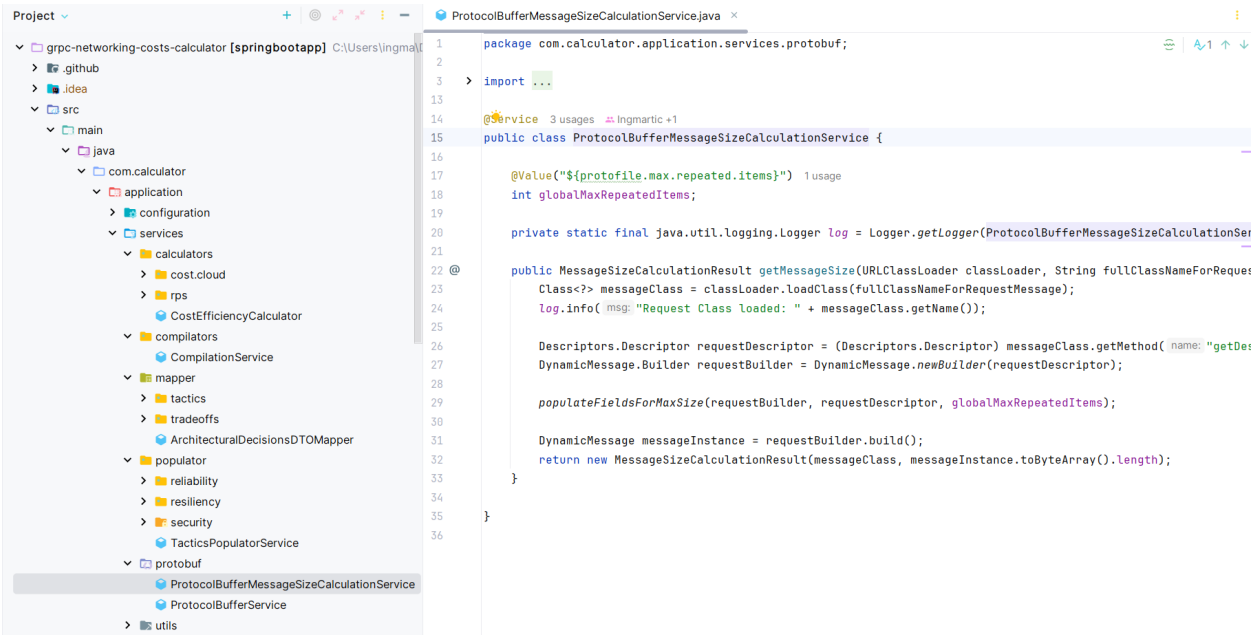


Figure 7.31: Protocol buffer services - Protocol Buffer IDL file message size calculator.

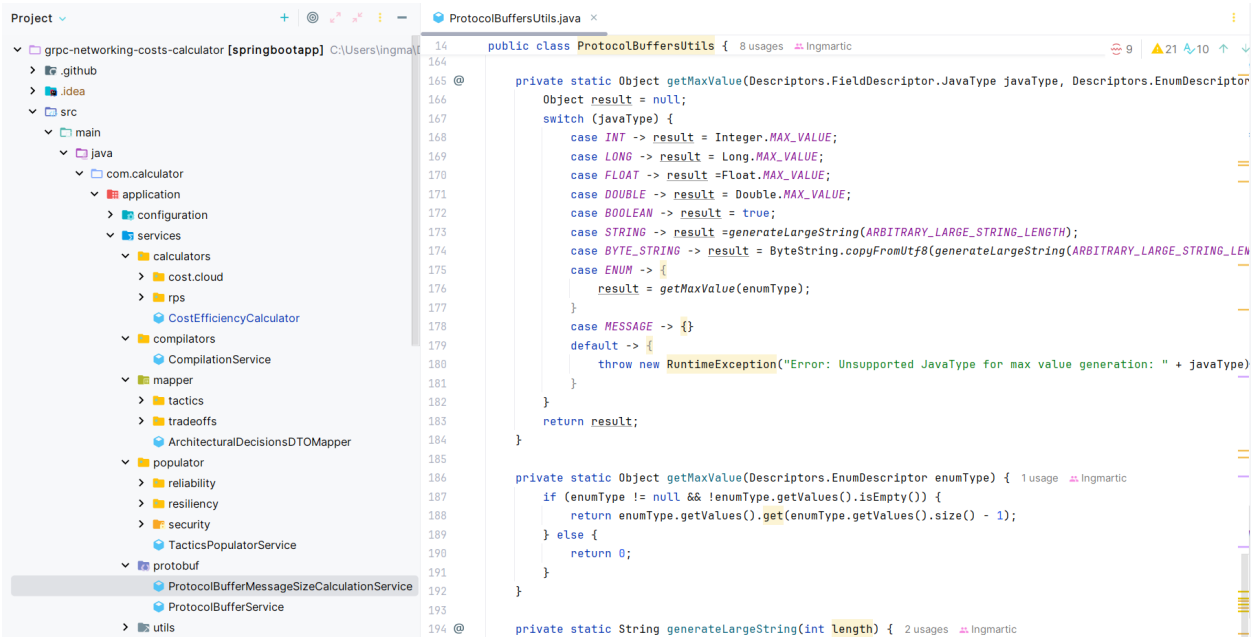


Figure 7.32: Value criteria to obtained for calculating the message response size for protocol buffer files.

Figure 7.33: Networking cost calculator AWS data transfer adapter.

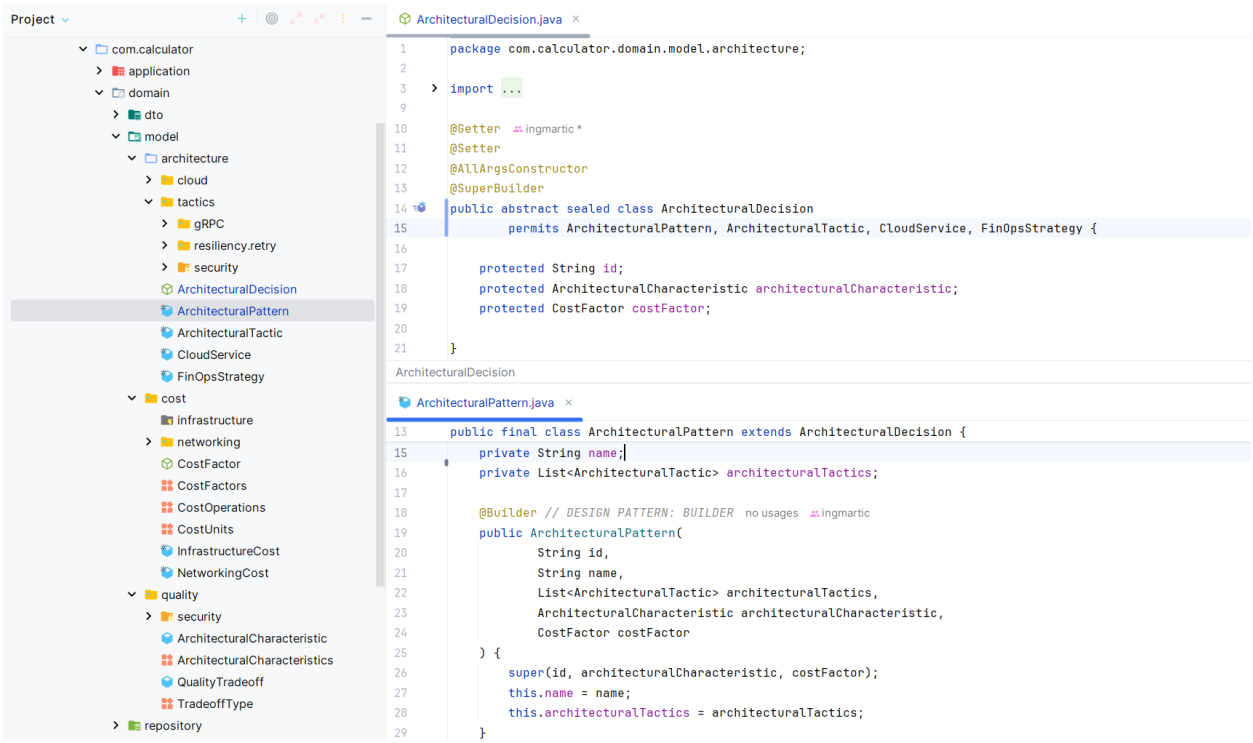


Figure 7.34: Domain model - Architectural decisions.

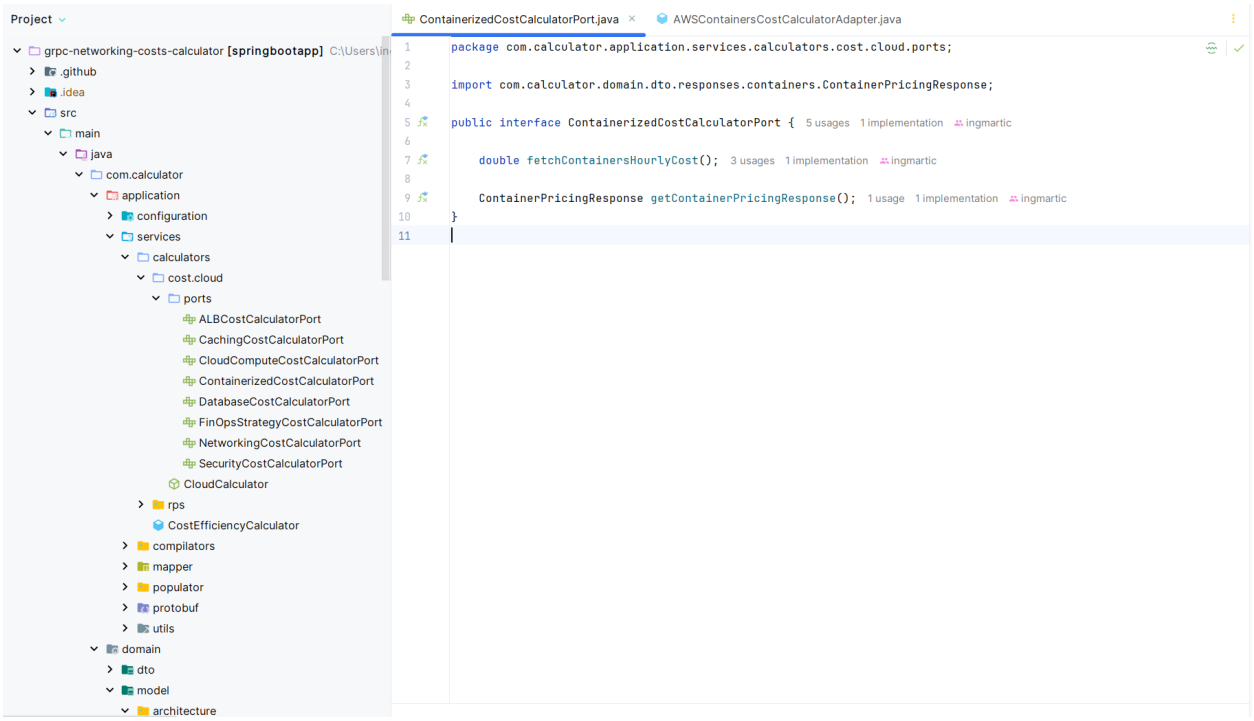


Figure 7.35: Hexagonal architecture ports - Containerized cost calculation port.

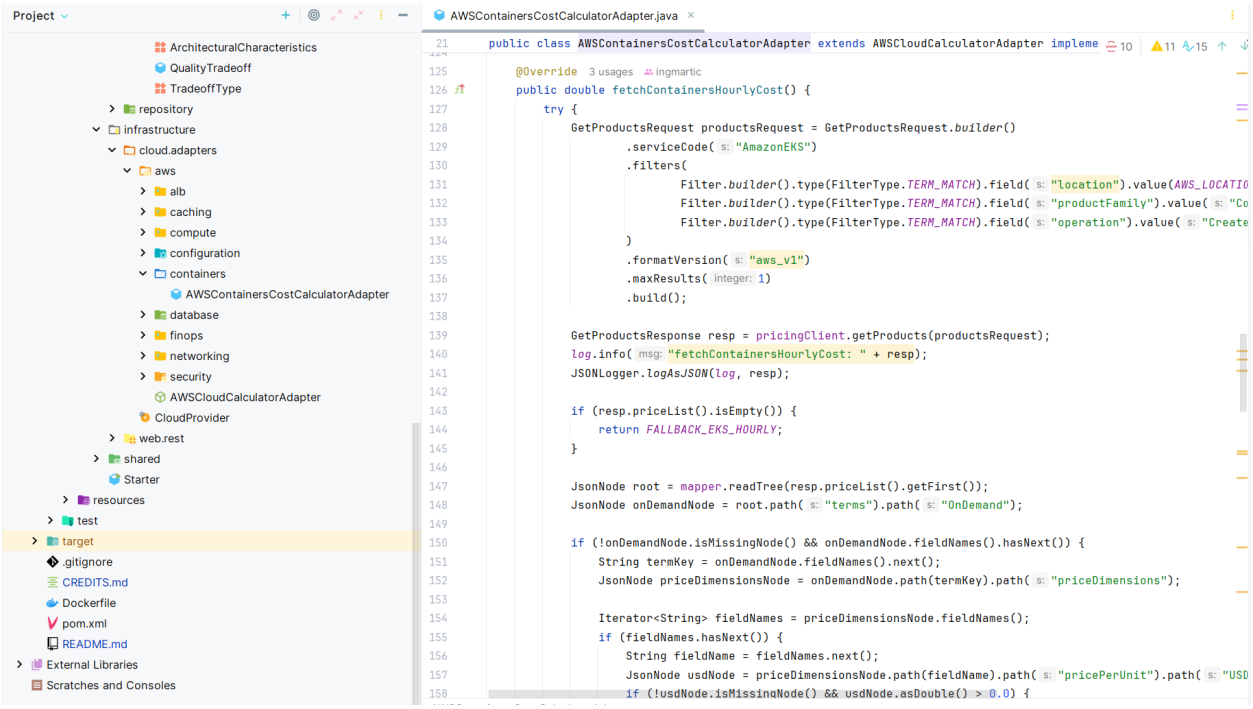


Figure 7.36: Hexagonal architecture adapters - Containers cost calculator adapter for AWS.

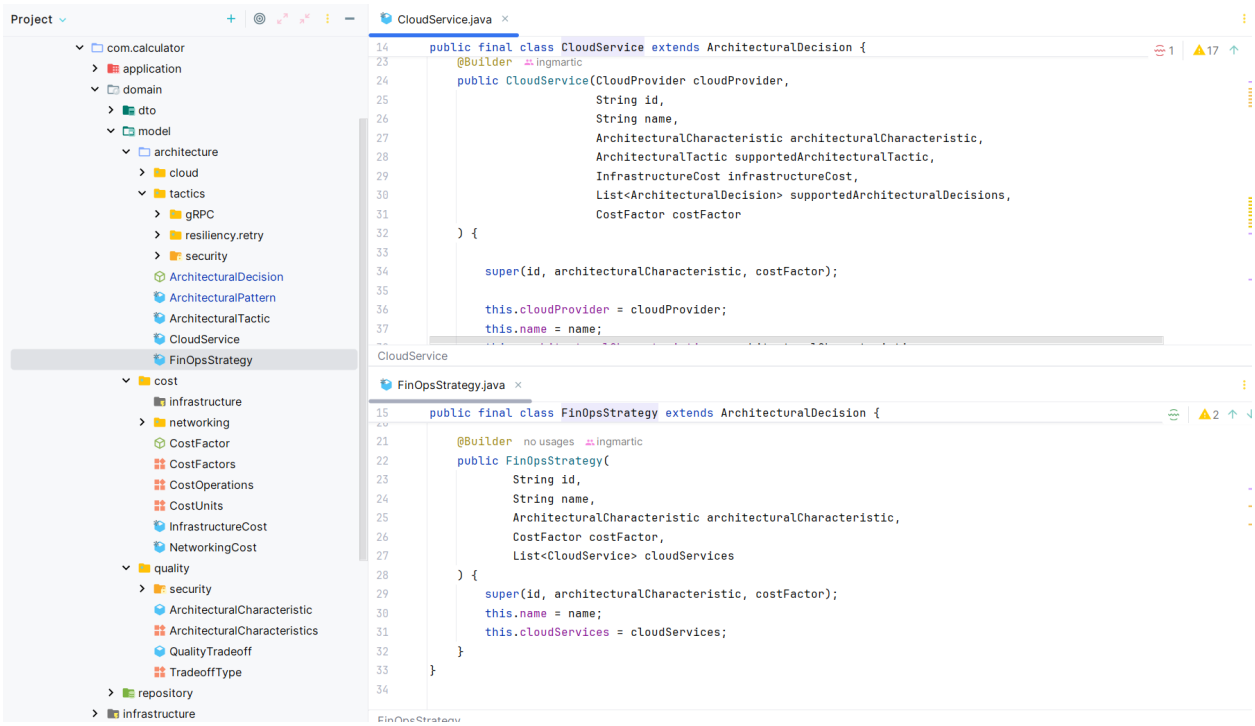


Figure 7.37: Domain model - Cloud service and FinOps strategy.

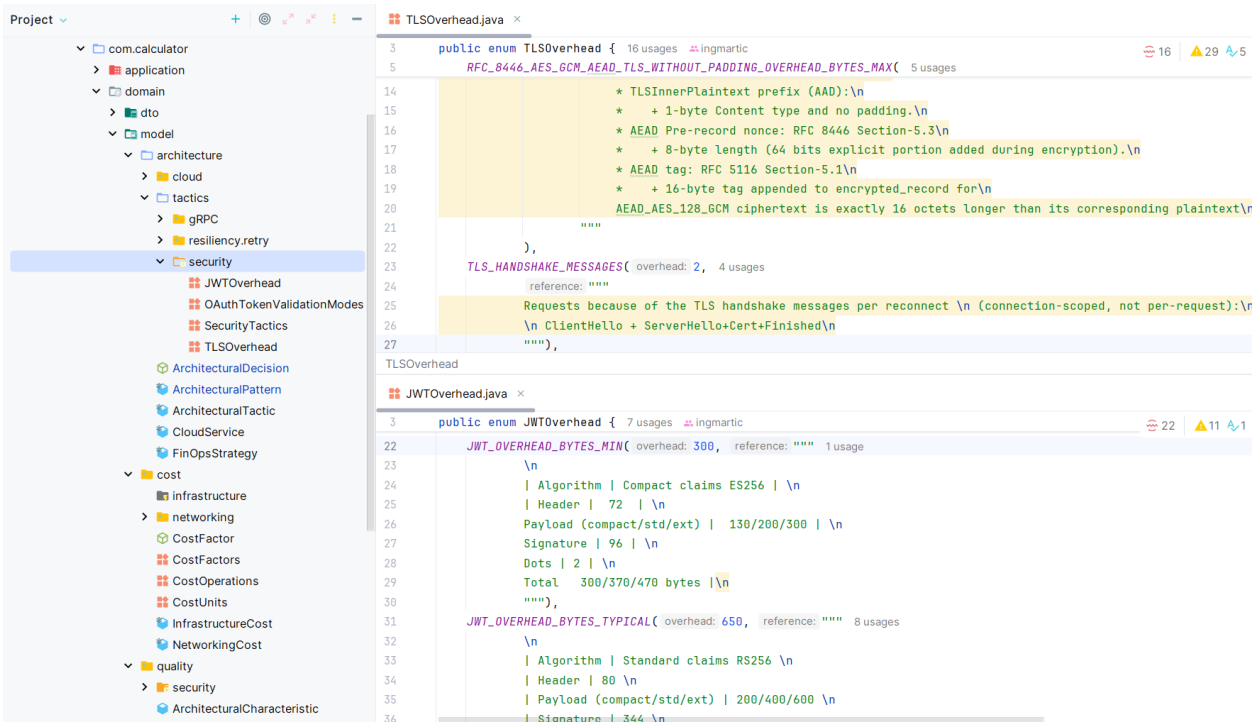


Figure 7.38: Domain model - Security tactics overhead.

SecurityArchitecturalDecisionRepository.java

```
20  public class SecurityArchitecturalDecisionRepository {                                  15
33      public ArchitecturalDecision getTLSTactic() {  10 usages  ingmartic
48              new SecurityQualityTradeoff(
49                      ArchitecturalCharacteristic.builder()
50                              .name(ArchitecturalCharacteristics.INTEGRITY.name())
51                              .build(),
52                      TradeoffType.PROMOTES,
53                      new String[]{"A02:2021","A07:2021"},
54                      new String[]{"Cryptographic Failures","Identification and Authentication Failures"},
55                      new String[]{"CWE-319","CWE-523","CWE-326"},
56                       vulnerabilityPrevented: "TLS 1.3 (RFC 8446) encrypts the gRPC channel in transit."
57              )
58          );
59          qualityTradeoffs.add(
60                  new QualityTradeoff(
61                          ArchitecturalCharacteristic.builder()
62                                  .name(ArchitecturalCharacteristics.AFFORDABILITY.name())
63                                  .build(),
64                          TradeoffType.INHIBITS
65                  )
66          );
67
68          return ArchitecturalTactic.builder()
69                  .id("tactic-tls")
70                  .name("TLS (One-way)")
71                  .architecturalCharacteristic(
72                          ArchitecturalCharacteristic.builder()
73                                  .name(ArchitecturalCharacteristics.SECURITY.name())
74                                  .qualityTradeoffs(qualityTradeoffs).build()
75                  ).costFactor(
76                          new NetworkingCost(
77                                  NetworkingCostCriteria.OVERHEAD,
78                                   costImpactNotes: "Adds ~" +
79                                          TLSOverhead.RFC_8446_AES_GCM_AEAD_TLS_WITHOUT_PADDING_OVERHEAD_BYTES_MAX.getOverhead()
80                                  + " B/frame TLS overhead. Reconnect RPS adds handshake requests."
```

Figure 7.39: Security tactics repository - TLS tactic.

ResiliencyArchitecturalDecisionRepository.java

```
20  public class ResiliencyArchitecturalDecisionRepository {
62      public ArchitecturalDecision getRetryPattern() {  2 usages  ingmartic
64          qualityTradeoffs.add(
65                  new QualityTradeoff(
66                          ArchitecturalCharacteristic.builder()
67                                  .name(ArchitecturalCharacteristics.RESILIENCY.name())
68                                  .build(),
69                          TradeoffType.PROMOTES
70                  )
71          );
72          qualityTradeoffs.add(
73                  new QualityTradeoff(
74                          ArchitecturalCharacteristic.builder()
75                                  .name(ArchitecturalCharacteristics.AFFORDABILITY.name())
76                                  .build(),
77                          TradeoffType.INHIBITS
78                  )
79          );
80
81          return ArchitecturalPattern.builder()
82                  .id("tactic-retry")
83                  .name("Retry")
84                  .architecturalCharacteristic(
85                          ArchitecturalCharacteristic.builder()
86                                  .name(ArchitecturalCharacteristics.RESILIENCY.name())
87                                  .qualityTradeoffs(qualityTradeoffs).build()
88                  ).costFactor(
89                          new NetworkingCost(
90                                  NetworkingCostCriteria.REQUESTS_PER_SECOND,
91                                  costImpactNotes: "Each retry adds to effective RPS. With a 10% error rate and 2 retries, " +
92                                          "effective RPS can increase by up to 20%."
93                          )
94                  )
95                  .build();
96      }
```

Figure 7.40: Resiliency tactics repository - Retry pattern.

ReliabilityArchitecturalDecisionRepository.java

```
18    public class ReliabilityArchitecturalDecisionRepository {
62        public ArchitecturalDecision getServerSideLoadBalancing() {  3 usages  ingmartic
63            List<QualityTradeoff> qualityTradeoffs = new ArrayList<>( initialCapacity: 1);
64            qualityTradeoffs.add(
65                    new QualityTradeoff(
66                            ArchitecturalCharacteristic.builder()
67                                    .name(ArchitecturalCharacteristics.RELIABILITY.name())
68                                    .build(),
69                            TradeoffType.PROMOTES
70                    )
71            );
72            qualityTradeoffs.add(
73                    new QualityTradeoff(
74                            ArchitecturalCharacteristic.builder()
75                                    .name(ArchitecturalCharacteristics.AFFORDABILITY.name())
76                                    .build(),
77                            TradeoffType.INHIBITS
78                    )
79            );
80
81            return ArchitecturalTactic.builder()
82                    .id("tactic-server-lb")
83                    .name("Server-side Load Balancing")
84                    .architecturalCharacteristic(
85                            ArchitecturalCharacteristic.builder()
86                                    .name(ArchitecturalCharacteristics.RELIABILITY.name())
87                                    .qualityTradeoffs(qualityTradeoffs).build()
88                    ).costFactor(
89                            new InfrastructureCost(
90                                    costImpactNotes: "Adds ALB fixed hourly charge + LCU costs from AWS Pricing API."
91                            )
92                    )
93                    .build();
94        }
```

Figure 7.41: Reliability tactics repository - Server-side load balancing tactic.

Figure 7.42: Cloud services repository - ELB/ALB Layer 7 supporting server-side load balancing.

CloudArchitecturalDecisionRepository.java

```java
public class CloudArchitecturalDecisionRepository {

    public ArchitecturalDecision getAmazonEKSControlPlaneCloudService() {  1 usage  ingmartic *

        List<QualityTradeoff> qualityTradeoffs = new ArrayList<>( initialCapacity: 2);
        qualityTradeoffs.add(new QualityTradeoff(
                ArchitecturalCharacteristic.builder()
                        .name(ArchitecturalCharacteristics.RELIABILITY.name())
                        .build(),
                TradeoffType.PROMOTES
        ));
        qualityTradeoffs.add(new QualityTradeoff(
                ArchitecturalCharacteristic.builder()
                        .name(ArchitecturalCharacteristics.AFFORDABILITY.name()).build(),
                TradeoffType.INHIBITS
        ));

        return CloudService.builder()
                .cloudProvider(CloudProvider.AWS)
                .id("service-eks-controlplane")
                .name("AWS EKS")
                .architecturalCharacteristic(
                        ArchitecturalCharacteristic.builder()
                                .name(ArchitecturalCharacteristics.RELIABILITY.name())
                                .qualityTradeoffs(qualityTradeoffs).build()
                ).costFactor(
                        new InfrastructureCost(
                                costImpactNotes: "Calculated via <a href=\"https://docs.aws.amazon.com/aws-cost-management/latest/APIReference/A
                        )
                )
                .build();
    }

}
```

Figure 7.43: Cloud services repository - AWS EKS.

FinOpsStrategyReliabilityArchitecturalDecisionRepository.java

```java
public class FinOpsStrategyReliabilityArchitecturalDecisionRepository {
    public ArchitecturalDecision getFinOpsStrategyForAWSApplicationLoadBalancer() { 4 usages  ingmartic *
        CloudService awsALB =
                (CloudService) cloudReliabilityArchitecturalDecisionRepository.getElasticLoadBalancerALBCloudService();

        List<QualityTradeoff> qualityTradeoffs = new ArrayList<>( initialCapacity: 1);
        qualityTradeoffs.add(
          new QualityTradeoff(
                  ArchitecturalCharacteristic.builder()
                          .name(ArchitecturalCharacteristics.AFFORDABILITY.name())
                          .build(),
                  TradeoffType.PROMOTES
          )
        );

        return FinOpsStrategy.builder()
                .id("finops-aws-alb")
                .name("FinOpsStrategy for ALB")
                .architecturalCharacteristic(
                        ArchitecturalCharacteristic.builder()
                                .name(ArchitecturalCharacteristics.AFFORDABILITY.name())
                                .qualityTradeoffs(qualityTradeoffs)
                                .build()
                )
                .cloudServices(List.of(awsALB))
                .costFactor(
                        new InfrastructureCost(
                                costImpactNotes: """
                                ALB has NO Reserved Instances or Savings Plans — only usage reduction cuts cost, \n
                                Consider NLB for pure TCP/UDP: NLCU pricing is often cheaper than ALB LCU at scale.
                                """
                        )
                )
                .build();
    }
```

Figure 7.44: Finops strategy repostory - spot instances for EC2 of the EKS cluster.

TCOUnitEconomicsController.java

```java

@RestController  1 usage  ingmartic *
@RequestMapping("/api/cost")
@CrossOrigin(origins = "*")
public class TCOUnitEconomicsController {

    private static final Logger log = Logger.getLogger(TCOUnitEconomicsController.class.getName());  4 usages

    @Autowired  2 usages
    private CostEfficiencyCalculator costEfficiencyCalculator;

    @PostMapping("/unit-economics")  no usages  ingmartic *
    public ResponseEntity<TCOUnitEconomicsResponse> calculateUnitEconomics(
            @RequestBody TCOUnitEconomicsRequest unitEconomicsRequest) {

        TCOUnitEconomicsResponse unitEconomicsResponse = costEfficiencyCalculator.calculateUnitEconomics(unitEconomicsRequest);
        unitEconomicsResponse.affordabilityImpact = costEfficiencyCalculator.summariseAffordabilityImpact().name();

        return ResponseEntity.ok(unitEconomicsResponse);
    }
```

Figure 7.45: TCO and UnitEconomics controller.

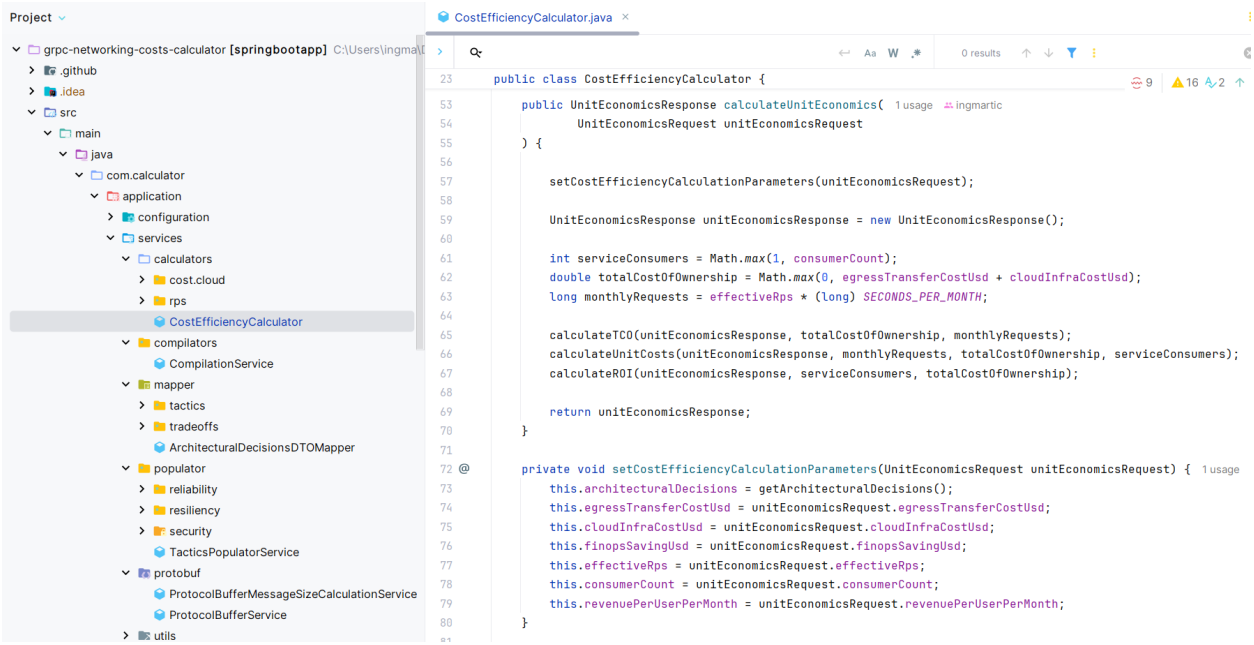


Figure 7.46: Cost-efficiency calculator method.

CostEfficiencyCalculator.java

0 results

```
23  public class CostEfficiencyCalculator {
82      private void calculateROI(UnitEconomicsResponse unitEconomicsResponse, int endUsers, double totalCostOfOwnership) {  1 usage  ingmartic *
84          unitEconomicsResponse.hasRevenueData = revenue > 0;
85
86          if (unitEconomicsResponse.hasRevenueData) {
87              double totalMonthlyRevenue = revenue * endUsers;
88              double netProfit = totalMonthlyRevenue - totalCostOfOwnership;
89
90              unitEconomicsResponse.totalMonthlyRevenueUsd = round2(totalMonthlyRevenue);
91              unitEconomicsResponse.totalAnnualRevenueUsd = round2( decimal: totalMonthlyRevenue * MONTHS_PER_YEAR);
92              unitEconomicsResponse.arpuMonthly = round2(revenue);
93
94              unitEconomicsResponse.monthlyRoiPct = totalCostOfOwnership > 0
95                      ? round2( decimal: netProfit / totalCostOfOwnership * 100) : 0;
96              unitEconomicsResponse.annualRoiPct = (totalCostOfOwnership > 0)
97                      ? round2( decimal: ((totalMonthlyRevenue * MONTHS_PER_YEAR)
98                      - (totalCostOfOwnership * MONTHS_PER_YEAR))
99                      / (totalCostOfOwnership * MONTHS_PER_YEAR) * 100)
100                     : 0;
101             unitEconomicsResponse.netMonthlyProfitUsd = round2(netProfit);
102             unitEconomicsResponse.netAnnualProfitUsd = round2( decimal: netProfit * MONTHS_PER_YEAR);
103             unitEconomicsResponse.breakEvenUsers = (revenue > 0 && totalCostOfOwnership > 0)
104                     ? (int) Math.ceil(totalCostOfOwnership / revenue) : 0;
105             unitEconomicsResponse.revenuePerDollarInfra = totalCostOfOwnership > 0
106                     ? round2( decimal: totalMonthlyRevenue / totalCostOfOwnership) : 0;
107             unitEconomicsResponse.netMarginPerUserMonthly = round4( decimal: revenue - unitEconomicsResponse.costPerUserPerMonthUsd);
108         }
109     }
```

Figure 7.47: Cost-efficiency calculator - calculate ROI method.

CostEfficiencyCalculator.java

```
23  public class CostEfficiencyCalculator {

126     public TradeoffType summariseAffordabilityImpact() {  1 usage  ingmartic
127         boolean anyInhibits = false;
128         boolean anyPromotes = false;
129
130         for (ArchitecturalDecision decision : architecturalDecisions) {
131             if (decision.getArchitecturalCharacteristic() == null) continue;
132             List<?> tradeoffs = decision.getArchitecturalCharacteristic().getQualityTradeoffs();
133             if (tradeoffs == null) continue;
134
135             for (Object tradeoff : tradeoffs) {
136                 if (!(tradeoff instanceof com.calculator.domain.model.quality.QualityTradeoff qt)) continue;
137                 if (qt.getArchitecturalCharacteristic() == null) continue;
138
139                 String attrName = qt.getArchitecturalCharacteristic().getName();
140                 if (!ArchitecturalCharacteristics.AFFORDABILITY.name().equalsIgnoreCase(attrName)) continue;
141
142                 TradeoffType type = qt.getTradeoffType();
143                 if (type == TradeoffType.INHIBITS)  anyInhibits = true;
144                 if (type == TradeoffType.PROMOTES)  anyPromotes = true;
145             }
146         }
147
148         if (anyInhibits) return TradeoffType.INHIBITS;
149         if (anyPromotes) return TradeoffType.PROMOTES;
150         return TradeoffType.ORTHOGONAL;
151     }
```

Figure 7.48: Cost-efficiency calculator - affordability impact summary method.

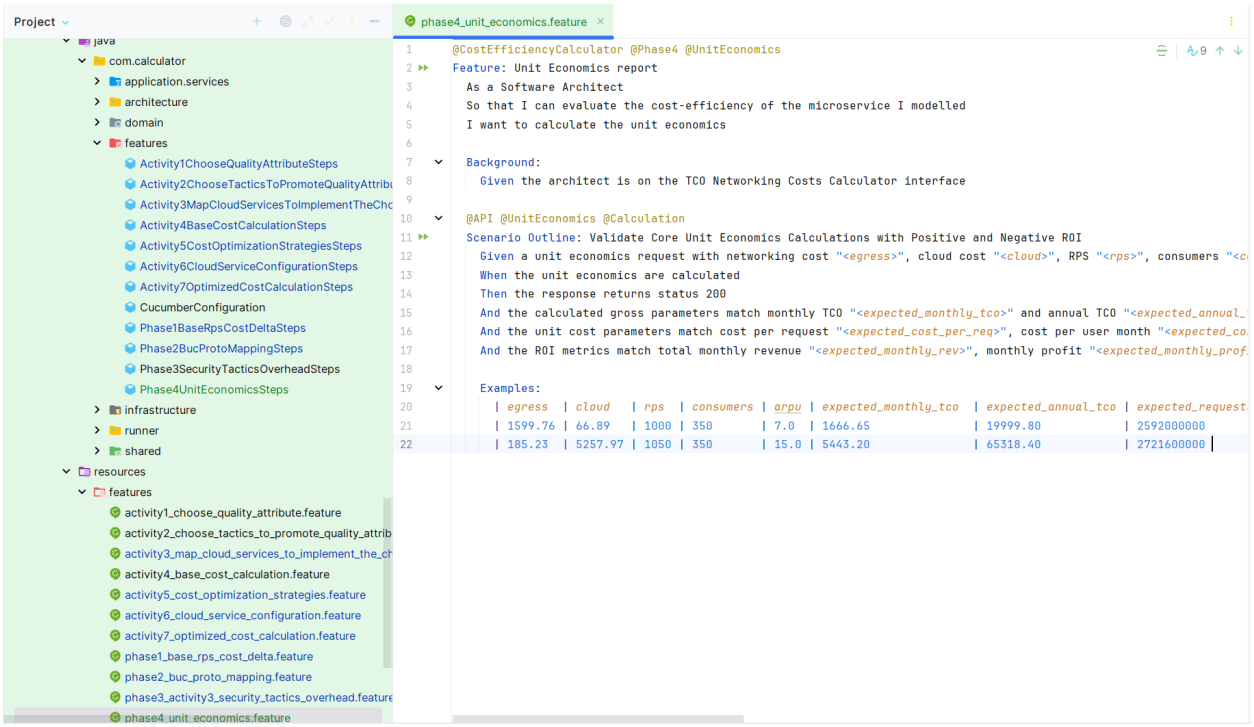


Figure 7.49: Phase 4 Unit economics feature cucumber file.

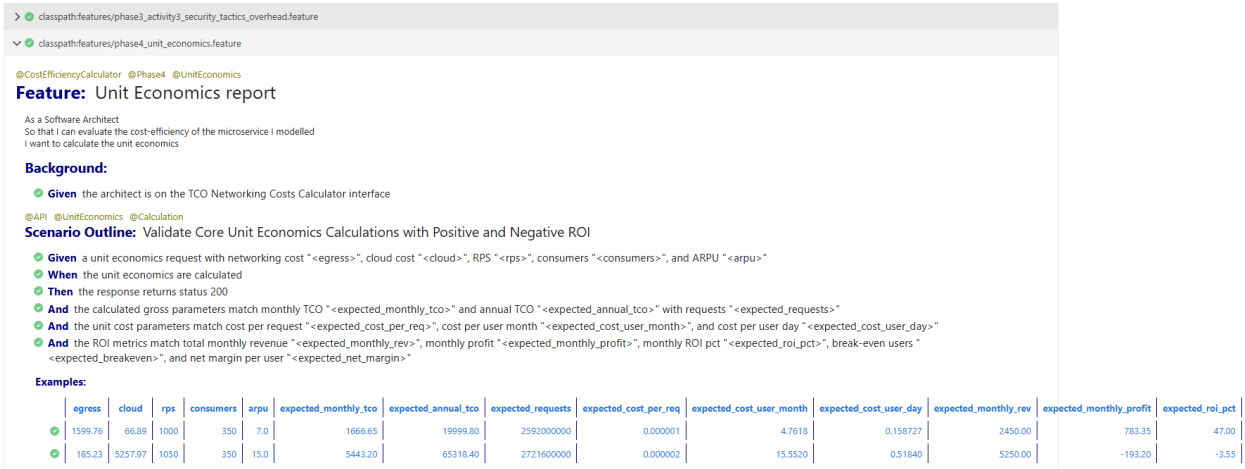


Figure 7.50: Cucumber tests report - Phase 4 Unit economics.

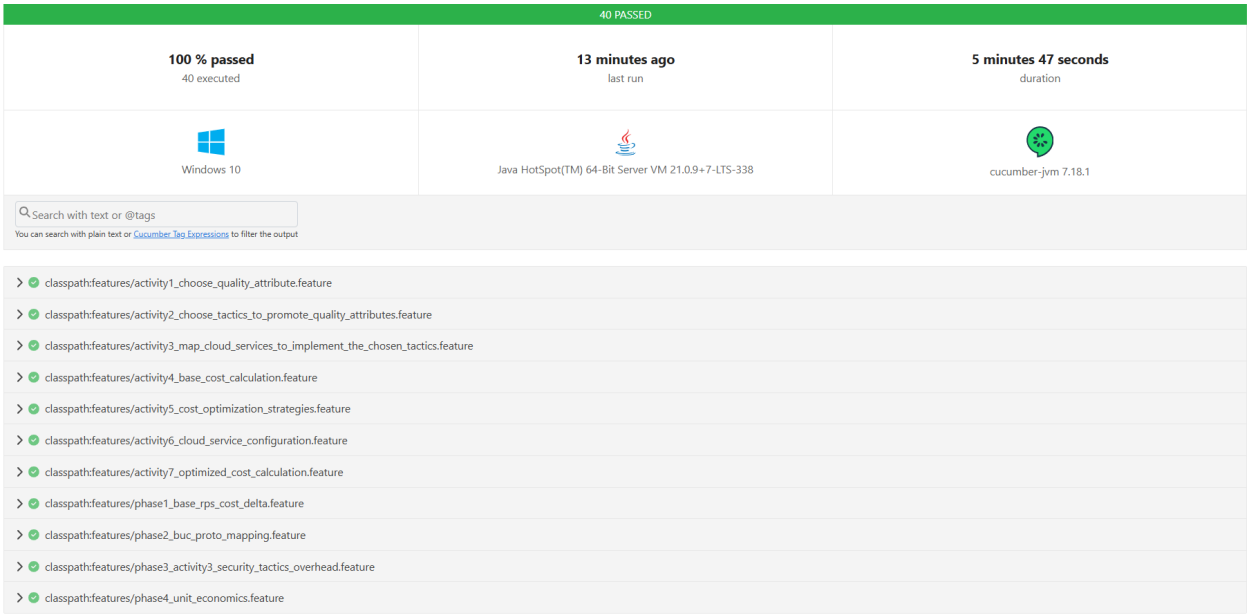


Figure 7.51: Cucumber tests report.

Bibliography

Adeel, T. (2025, 12). *gRPC in 2025: Why Top Companies Are Switching from REST | by Talha Adeel | Medium.* Retrieved from https://medium.com/@miantalha.t08/grpc-in-2025-why-top-companies-are-switching-from-rest-36e3c6e2ec4c

Adzic, G. (2011). *Specification By Example How successful teams deliver the right software.* MANNING.

Adzic, G. (2020, 3). *Specification by Example, 10 years later.* Retrieved from https://gojko.net/2020/03/17/sbe-10-years.html

Amazon Web Services. (2026). *bcm-pricing-calculator — AWS CLI 2.35.7 Command Reference.* Retrieved from https://docs.aws.amazon.com/cli/latest/reference/bcm-pricing-calculator/

Amazon Web Services, I. (2019). *AWS Well-Architected Labs :: AWS Well-Architected Labs.* Retrieved from https://www.wellarchitectedlabs.com/

Amazon Web Services, I. (2021). *Karpenter.* Retrieved from https://karpenter.sh/

Amazon Web Services, I. (2023a, 10). *AWS Well-Architected Framework.* Retrieved from https://docs.aws.amazon.com/wellarchitected/latest/framework/welcome.html

Amazon Web Services, I. (2023b, 10). *Definitions.* Retrieved from https://docs.aws.amazon.com/wellarchitected/latest/framework/definitions.html

Amazon Web Services, I. (2023c, 4). *Failure management - AWS Well-Architected Framework (2023-04-10).* Retrieved from https://docs.aws.amazon.com/wellarchitected/2023-04-10/framework/a-failure-management.html

Amazon Web Services, I. (2023d, 10). *Resilience lifecycle framework: A continuous approach to resilience improvement.*

Amazon Web Services, I. (2024a, 11). *AWS Well-Architected Framework - AWS Well-Architected Framework.* Retrieved from https://docs.aws.amazon.com/wellarchitected/latest/framework/welcome.html

Amazon Web Services, I. (2024b, 6). *Cost Optimization Pillar - AWS Well-Architected Framework - Cost Optimization Pillar.* Retrieved from https://docs.aws.amazon.com/wellarchitected/latest/cost-optimization-pillar/welcome.html

Amazon Web Services, I. (2024c). *Distribución del tráfico de aplicaciones y de red - Precios de Elastic Load Balancing - AWS.* Retrieved from https://aws.amazon.com/es/elasticloadbalancing/pricing/

Amazon Web Services, I. (2024d, 11). *Security Pillar - AWS Well-Architected Framework - Security Pillar.* Retrieved from https://docs.aws.amazon.com/wellarchitected/latest/security-pillar/welcome.html

Amazon Web Services, I. (2024e). *What are Microservices?* Retrieved from https://aws.amazon.com/microservices/

Amazon Web Services, I. (2024f). *What is Cloud Native?* Retrieved from https://aws.amazon.com/what-is/cloud-native/

Amazon Web Services, I. (2025a). *AWS Pricing Calculator.* Retrieved from https://calculator.aws/#/

Amazon Web Services, I. (2025b). *COST01-BP02 Establish a partnership between finance and technology.* Retrieved from https://docs.aws.amazon.com/wellarchitected/latest/framework/cost_cloud_financial_management_partnership.html

Amazon Web Services, I. (2025c). *COST01-BP03 Establish cloud budgets and forecasts.* Retrieved from https://docs.aws.amazon.com/wellarchitected/latest/framework/cost_cloud_financial_management_budget_forecast.html

Amazon Web Services, I. (2025d). *COST01-BP04 Implement cost awareness in your organizational processes.* Retrieved from https://docs.aws.amazon.com/wellarchitected/latest/framework/cost_cloud_financial_management_cost_awareness.html

Amazon Web Services, I. (2025e). *COST01-BP05 Report and notify on cost optimization.* Retrieved from https://docs.aws.amazon.com/wellarchitected/latest/framework/cost_cloud_financial_management_usage_report.html

Amazon Web Services, I. (2025f). *COST01-BP09 Quantify business value from cost optimization.* Retrieved from https://docs.aws.amazon.com/wellarchitected/latest/framework/cost_cloud_financial_management_quantify_value.html

Amazon Web Services, I. (2025g). *COST05-BP01 Identify organization requirements for cost.* Retrieved from https://docs.aws.amazon.com/wellarchitected/latest/framework/cost_select_service_requirements.html

Amazon Web Services, I. (2025h). *COST05-BP05 Select components of this workload to optimize cost in line with organization priorities.* Retrieved from https://docs.aws.amazon.com/wellarchitected/latest/framework/cost_select_service_select_for_cost.html

Amazon Web Services, I. (2025i). *COST05-BP06 Perform cost analysis for different usage over time.* Retrieved from https://docs.aws.amazon.com/wellarchitected/latest/framework/cost_select_service_analyze_over_time.html

Amazon Web Services, I. (2025j). *COST06-BP01 Perform cost modeling.* Retrieved from https://docs.aws.amazon.com/wellarchitected/latest/framework/cost_type_size_number_resources_cost_modeling.html

Amazon Web Services, I. (2025k). *COST06-BP02 Select resource type, size, and number based on data.* Retrieved from https://docs.aws.amazon.com/wellarchitected/latest/framework/cost_type_size_number_resources_data.html

Amazon Web Services, I. (2025l). *COST07-BP04 Implement pricing models for all components of this workload.* Retrieved from https://docs.aws.amazon.com/wellarchitected/latest/framework/cost_pricing_model_implement_models.html

Amazon Web Services, I. (2025m). *COST07-BP05 Perform pricing model analysis at the management account level.* Retrieved from https://docs.aws.amazon.com/wellarchitected/latest/framework/cost_pricing_model_master_analysis.html

Amazon Web Services, I. (2025n). *COST09-BP01 Perform an analysis on the workload demand - AWS Well-Architected Framework.* Retrieved from https://docs.aws.amazon.com/wellarchitected/latest/framework/cost_manage_demand_resources_cost_analysis

.html

Amazon Web Services, I. (2025o). *COST09-BP02 Implement a buffer or throttle to manage demand - AWS Well-Architected Framework.* Retrieved from https://docs.aws.amazon.com/wellarchitected/latest/framework/cost_manage_demand_resources_buffer_throttle.html

Amazon Web Services, I. (2025p). *Cost-effective resources.* Retrieved from https://docs.aws.amazon.com/wellarchitected/latest/framework/a-cost-effective-resources.html

Amazon Web Services, I. (2025q). *Cost optimization.* Retrieved from https://docs.aws.amazon.com/wellarchitected/latest/framework/cost-optimization.html

Amazon Web Services, I. (2025r). *Cost Optimization Design Principles.* Retrieved from https://docs.aws.amazon.com/wellarchitected/latest/framework/cost-dp.html

Amazon Web Services, I. (2025s, 10). *Introducing Amazon Quick Suite: your agentic AI-powered workspace - AWS.* Retrieved from https://aws.amazon.com/about-aws/whats-new/2025/10/amazon-quick-suite-agentic-ai-powered-workspace/

Amazon Web Services, I. (2025t). *Manage demand and supply resources - AWS Well-Architected Framework.* Retrieved from https://docs.aws.amazon.com/wellarchitected/latest/framework/a-manage-demand-and-supply-resources.html

Amazon Web Services, I. (2025u). *Practice Cloud Financial Management.* Retrieved from https://docs.aws.amazon.com/wellarchitected/latest/framework/cost-cfm.html

Amazon Web Services, I. (2025v). *Practice Cloud Financial Management - Cost Optimization Pillar.* Retrieved from https://docs.aws.amazon.com/wellarchitected/latest/cost-optimization-pillar/practice-cloud-financial-management.html

Amazon Web Services, I. (2026a). *Amazon CloudWatch Pricing | Free Tier Available.* Retrieved from https://aws.amazon.com/cloudwatch/pricing/

Amazon Web Services, I. (2026b). *Amazon EC2 instance network bandwidth - Amazon Elastic Compute Cloud.* Retrieved from https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-network-bandwidth.html

Amazon Web Services, I. (2026c). *Amazon EC2 T3 Instances – Amazon Web Services (AWS).* Retrieved from https://aws.amazon.com/ec2/instance-types/t3/

Amazon Web Services, I. (2026d). *Amazon EKS Pricing.* Retrieved from https://aws.amazon.com/eks/pricing/

Amazon Web Services, I. (2026e). *Certificate Manager - AWS Certificate Manager - AWS.* Retrieved from https://aws.amazon.com/certificate-manager/

Amazon Web Services, I. (2026f). *COST07-BP02 Choose Regions based on cost - Cost Optimization Pillar.* Retrieved from https://docs.aws.amazon.com/wellarchitected/latest/cost-optimization-pillar/cost_pricing_model_region_cost.html

Amazon Web Services, I. (2026g). *Data protection - AWS Well-Architected Framework (2023-04-10).* Retrieved from https://docs.aws.amazon.com/wellarchitected/2023-04-10/framework/a-data-protection.html

Amazon Web Services, I. (2026h). *Design principles - Reliability Pillar.* Retrieved from https://docs.aws.amazon.com/wellarchitected/latest/reliability-pillar/

design-principles.html

Amazon Web Services, I. (2026i). *Karpenter - Amazon EKS.* Retrieved from https://docs.aws.amazon.com/eks/latest/best-practices/karpenter.html

Amazon Web Services, I. (2026j). *PricingClient (AWS SDK for Java - 2.48.3).* Retrieved from https://docs.aws.amazon.com/java/api/latest/software/amazon/awssdk/services/pricing/PricingClient.html

Amazon Web Services, I. (2026k). *REL11-BP02 Fail over to healthy resources - Reliability Pillar.* Retrieved from https://docs.aws.amazon.com/wellarchitected/latest/reliability-pillar/rel_withstand_component_failures_failover2good.html

Amazon Web Services, I. (2026l). *Reliability - Reliability Pillar.* Retrieved from https://docs.aws.amazon.com/wellarchitected/latest/reliability-pillar/reliability.html

Amazon Web Services, I. (2026m). *Resiliency, and the components of reliability - Reliability Pillar.* Retrieved from https://docs.aws.amazon.com/wellarchitected/latest/reliability-pillar/resiliency-and-the-components-of-reliability.html

Amazon Web Services, I. (2026n). *SEC09-BP01 Implement secure key and certificate management - AWS Well-Architected Framework (2023-04-10).* Retrieved from https://docs.aws.amazon.com/wellarchitected/2023-04-10/framework/sec_protect_data_transit_key_cert_mgmt.html

Amazon Web Services, I. (2026o). *SEC09-BP02 Enforce encryption in transit - AWS Well-Architected Framework (2023-04-10).* Retrieved from https://docs.aws.amazon.com/wellarchitected/2023-04-10/framework/sec_protect_data_transit_encrypt.html

Amazon Web Services, I. (2026p). *Security foundations - Security Pillar.* Retrieved from https://docs.aws.amazon.com/wellarchitected/latest/security-pillar/security.html

Amazon Web Services, I. (2026q). *Specifications for Amazon EC2 general purpose instances - Amazon EC2.* Retrieved from https://docs.aws.amazon.com/ec2/latest/instancetypes/gp.html

Amazon Web Services, I. (2026r). *Understanding data transfer charges - AWS Data Exports.* Retrieved from https://docs.aws.amazon.com/cur/latest/userguide/cur-data-transfers-charges.html

Amazon Web Services, I. (2026s). *Update the Availability Zones for your Application Load Balancer - Elastic Load Balancing.* Retrieved from https://docs.aws.amazon.com/elasticloadbalancing/latest/application/load-balancer-subnets.html

Amstrong, J. (2020). *Migrating to AWS: A Manager's Guide* (1st Edition ed.). O'Reilly.

Anthropic. (2026). *Claude Code overview - Claude Code Docs.* Retrieved from https://code.claude.com/docs/en/overview

Avolio, C. (2023). *Introduzione allo studio del Dialetto Siciliano* (1st ed.). outlook.

Babal, H. (2024). *gRPC Microservices in Go.* Manning.

Bass, L. (2025). *Engineering AI Systems: Architecture and DevOps Essentials.* Retrieved from https://learning.oreilly.com/library/view/engineering-ai-systems/9780138261542/

Bass, L., Clements, P., & Kazman, R. (2021). *Software Architecture in Practice (SEI Series in*

Software Engineering) (4th edition ed.).

Beck, K. (2002). *Test Driven Development: By Example* (1st Edition ed.).

Betts, T. (2023, 4). *InfoQ Software Architecture and Design Trends Report - April 2023.*

Block, Hirt, & Danielsen. (2017). *Foundations of Financial Management* (Sixteenth edition ed.). McGrawHill Education.

Borowiec, R. (2026). *Spring MVC and Thymeleaf: how to access data from templates - Thymeleaf.* Retrieved from https://www.thymeleaf.org/doc/articles/springmvcaccessdata.html

Brandhorst-Satzkorn, J. (2019, 5). *Support gRPC over HTTP/3.* Retrieved from https://github.com/grpc/grpc/issues/19126

Broadcom. (2026). *17. Web MVC framework.* Retrieved from https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html

Brown, S. (2025). *The C4 Model for Visualizing Software Architecture.* Retrieved from https://leanpub.com/visualising-software-architecture

Brown, S. (2026). *Home | Structurizr.* Retrieved from https://docs.structurizr.com/

Callanan, M. (2024, 3). *Cloud FinOps & Kubernetes Optimisation at Scale • Matt Callanan • YOW! 2023.* Retrieved from https://www.youtube.com/watch?v=_F12WgQuTI8

Carnell, J., & Huaylupo, I. (2021). *Spring Microservices in Action* (Second Edition ed.). Manning.

Cervantes, R., Humberto; Kazman. (2016, 5). *Designing Software Architectures: A Practical Approach.* Retrieved from https://learning.oreilly.com/library/view/designing-software-architectures/9780134390857/

Chávez, W. (2026). *grpc-networking-costs-calculator.*

Chávez González, W. M. (2025). *UniversidadJaveriana-MastersDegreeTeam/end-2-end-model-finops-grpc-architects-tool-service: end-2-end-model-finops-grpc-architects-tool-service.* Retrieved from https://github.com/UniversidadJaveriana-MastersDegreeTeam/end-2-end-model-finops-grpc-architects-tool-service

Chris Richardson Consulting, I. (2025, 5). *Chris Richardson Consulting, Inc - Distributed Data Patterns for Microservices - Modules.* Retrieved from https://microservices.matrixlms.com/learner_modules/list/350821

Chung, G. (2023, 2). *Using AWS tools to implement your Cloud Financial Management strategy.*

Chung, P. (2022). *AWS FinOps Simplified Eliminate cloid waste through practical FinOps* (First Edition ed.). Packt Publishing Ltd.

Ciceri, C., Farley, D., Ford, N., Harmel-Law, A., Keeling, M., Lilienthal, C., … Woods, E. (2022). *Software Architecture Metrics* (1st Edition ed.). O'Reilly.

Clements, L. B. F., Paul; Bass. (2011). *Documenting Software Architectures: Views and Beyond, Second Edition.* Pearson Education, Inc. Retrieved from https://learning.oreilly.com/library/view/documenting-software-architectures/9780132488617/

Cloud Native Computing Foundation. (2026a). *Graduated and Incubating Projects | CNCF.* Retrieved from https://www.cncf.io/projects/

Cloud Native Computing Foundation. (2026b). *Who We Are | CNCF.* Retrieved from https://www.cncf.io/about/who-we-are/

Cockburn, A. (2005, 9). *hexagonal-architecture.* Retrieved from https://alistair.cockburn.us/

hexagonal-architecture

Credly. (2023, 12). *AWS Certified Cloud Practitioner - Credly.* Retrieved from https://www.credly.com/badges/e1a0f521-1dc4-42d1-b291-2dafd42002d2

Credly. (2024a, 7). *FinOps Certified Engineer - Credly.* Retrieved from https://www.credly.com/badges/5b83aca2-6b3e-4794-a783-f7fb4481c7d4

Credly. (2024b, 3). *gRPC [Java] Master Class - Credly.* Retrieved from https://www.credly.com/badges/805ce298-3efa-4ecc-b6a4-99963cd2a2bf

Credly. (2025, 6). *Intro to Total Cost of Ownership (TCO) | rideMACH - Credly.* Retrieved from https://www.credly.com/badges/c24e312c-4d03-4bf5-bc9a-77b8a91f7320

Credly. (2026, 2). *Cloud GreenOps Philosopher - Credly.* Retrieved from https://www.credly.com/badges/fe8e7edc-0c60-420a-8317-386e58946649

Doerr, J. (2019). *Measure What Matters* (1st ed.). Penguin Publishing Group.

Eldh, S. (2026, 3). Why You Should Care About Green Clean Sustainable Software. *IEEE Software*, *43*(02), 4–7. Retrieved from https://www. doi: 10.1109/MS.2025.3646318

Eliot, S., & Valverde, L. (2018, 5). *The 6 Pillars of the AWS Well-Architected Framework.*

Emer, B., Richey, C., Harvey, E., & Barto, J. (2023, 10). *Resilience lifecycle framework: A continuous approach to resilience improvement - AWS Prescriptive Guidance.* Retrieved from https://docs.aws.amazon.com/prescriptive-guidance/latest/resilience-lifecycle-framework/introduction.html

Erder, P. W. E., Murat; Pureur. (2021). *Continuous Architecture in Practice: Software Architecture in the Age of Agility and DevOps.* Retrieved from https://learning.oreilly.com/library/view/continuous-architecture-in/9780136523796/

Fernando, J. (2026, 2). *What Is Return on Investment (ROI) and How to Calculate It.* Retrieved from https://www.investopedia.com/terms/r/returnoninvestment.asp

FinOps Foundation. (2025, 10). *Introduction to Cloud Unit Economics.* Retrieved from https://www.finops.org/wg/introduction-cloud-unit-economics/

FinOps Foundation Project a Series of LF Projects, L. (2022). *2022 State of FinOps.* Retrieved from https://data.finops.org/

FinOps Foundation Project a Series of LF Projects, L. (2024a). *Architecting for Cloud.* Retrieved from https://www.finops.org/framework/capabilities/architecting-for-cloud/

FinOps Foundation Project a Series of LF Projects, L. (2024b). *FinOps Certified Engineer.* Retrieved from https://learn.finops.org/path/finops-certified-engineer

FinOps Foundation Project a Series of LF Projects, L. (2024c). *Guide to Cloud Service Provider Tools and Terminology.* Retrieved from https://www.finops.org/wg/multi-cloud-tools-and-terminology/

FinOps Foundation Project a Series of LF Projects, L. (2024d). *Introduction to FinOps.* Retrieved from https://learn.finops.org/introduction-to-finops

FinOps Foundation Project a Series of LF Projects, L. (2024e). *Introduction to FOCUS.* Retrieved from https://learn.finops.org/introduction-to-focus

FinOps Foundation Project a Series of LF Projects, L. (2024f). *Why Cost-Aware Architecture and a Reliable Data Lake Are Critical to FinOps Success.* Retrieved from https://www.youtube

.com/watch?v=AiOJB05qZmw

FinOps Foundation Project a Series of LF Projects, L. (2025a). *Architecting for Cloud FinOps Framework Capability.* Retrieved from https://www.finops.org/framework/capabilities/architecting-for-cloud/

FinOps Foundation Project a Series of LF Projects, L. (2025b). *Budgeting FinOps Framework Capability.* Retrieved from https://www.finops.org/framework/capabilities/budgeting/

FinOps Foundation Project a Series of LF Projects, L. (2025c). *Calculating Container Costs.* Retrieved from https://www.finops.org/wg/calculating-container-costs/

FinOps Foundation Project a Series of LF Projects, L. (2025d). *Capability: Unit Economics.* Retrieved from https://www.finops.org/framework/capabilities/unit-economics/

FinOps Foundation Project a Series of LF Projects, L. (2025e). *Domain: Manage the FinOps Practice.* Retrieved from https://www.finops.org/framework/domains/manage-finops-practice/

FinOps Foundation Project a Series of LF Projects, L. (2025f). *Domain: Optimize Usage & Cost.* Retrieved from https://www.finops.org/framework/domains/optimize-usage-cost/

FinOps Foundation Project a Series of LF Projects, L. (2025g). *Domain: Quantify Business Value.* Retrieved from https://www.finops.org/framework/domains/quantify-business-value/

FinOps Foundation Project a Series of LF Projects, L. (2025h). *FinOps Framework Overview.* Retrieved from https://www.finops.org/framework/

FinOps Foundation Project a Series of LF Projects, L. (2025i). *FinOps Personas.* Retrieved from https://www.finops.org/framework/personas/

FinOps Foundation Project a Series of LF Projects, L. (2025j). *FinOps Principles.* Retrieved from https://www.finops.org/framework/principles/

FinOps Foundation Project a Series of LF Projects, L. (2025k). *Rate Optimization FinOps Framework Capability.* Retrieved from https://www.finops.org/framework/capabilities/rate-optimization/

Firesmith, D. (2009). *The Method Framework for Engineering System Architectures*. CRC Press.

Firesmith, D. (2019a, 12). *System Resilience Part 2: How System Resilience Relates to Other Quality Attributes.* Retrieved from https://insights.sei.cmu.edu/blog/system-resilience-part-2-how-system-resilience-relates-to-other-quality-attributes/

Firesmith, D. (2019b, 11). *System Resilience: What Exactly is it?* Retrieved from https://insights.sei.cmu.edu/blog/system-resilience-what-exactly-is-it/

Firesmith, D. (2020, 4). *System Resilience Part 7: 16 Guiding Principles for System Resilience.* Retrieved from https://insights.sei.cmu.edu/blog/system-resilience-part-7-16-guiding-principles-for-system-resilience/

Ford, N., Richards, M., Sadalage, P., & Dehghani, Z. (2021). *Software Architecture: The Hard Parts* (1st Edition ed.). O'Reilly.

Ghanizada, I. (2021). *Google Cloud Certified Professional Cloud Architect* (1st ed.; R. Foltak, Ed.). Mc Graw Hill.

Google. (2022). *Protocol Buffers v3.* Retrieved from https://cloud.google.com/apis/design/

proto3?hl=es-419

Grande, A. (2025, 5). *EKS Pricing: A Complete Breakdown (2025 Guide) | DevZero.* Retrieved from https://www.devzero.io/blog/eks-pricing

gRPC Authors. (2024). *Authentication | gRPC.* Retrieved from https://grpc.io/docs/guides/auth/

Hohpe, G., & Woolf, B. (2004). *Enterprise Integration Patterns* (1st Edition ed.). Addison Wesley.

Holiday, R. (2014). The obstacle is the way : the timeless art of turning trials into triumph. , xiv.

Indrasiri, K., & Kuruppu, D. (2020). *gRPC: Up and Running: Building Cloud Native Applications with Go and Java for Docker and Kubernetes* (1st ed.). O'Reilly.

Indrasiri, K., & Suhothayan, S. (2021). *Design Patterns for Cloud Native Applications: Patterns in Practice Using APIs, Data, Events, and Streams* (1st ed.). O'Reilly.

ISO/IEC JTC 1/SC 7. (2023a). *ISO/IEC 25010:2023 Systems and software engineering-Systems and software Quality Requirements and Evaluation (SQuaRE)-Product quality model* (Tech. Rep.). Retrieved from www.iso.org

ISO/IEC JTC 1/SC 7. (2023b). *ISO/IEC 25019:2023.* Retrieved from https://www.iso.org/standard/78177.html

Joint Development Foundation Projects, G. S. F. S., LLC. (2026a). *Architecture | Green Software Patterns.* Retrieved from https://patterns.greensoftware.foundation/architecture

Joint Development Foundation Projects, G. S. F. S., LLC. (2026b). *Green Software Patterns.* Retrieved from https://patterns.greensoftware.foundation/

Jones, M., Bradley, J., & Sakimura, N. (2015, 5). JSON Web Signature (JWS). Retrieved from https://www.rfc-editor.org/info/rfc7515 doi: 10.17487/RFC7515

Jones, M., Bradley, J., Sakimura, N., Jones, M., Bradley, J., & Sakimura, N. (2015, 5). JSON Web Token (JWT). Retrieved from https://www.rfc-editor.org/info/rfc7519/ doi: 10.17487/RFC7519

Kenton, W. (2026, 4). *Net Income: Definition, Calculation, and Business Impact.* Retrieved from https://www.investopedia.com/terms/n/netincome.asp

Khanin, T., Daniil; Zharova. (2024, 3). *Unit economics: Data Driven Decisions for Business and Startups.* Retrieved from https://www.amazon.com/dp/B0CZ59CPRL?lv=shuf&channelId=500&plpRedirect=mhFallback

Kubernetes. (2025). *Vertical Pod Autoscaler.* Retrieved from https://github.com/kubernetes/autoscaler/tree/master/vertical-pod-autoscaler

Kubernetes. (2026). *Horizontal Pod Autoscaling | Kubernetes.* Retrieved from https://kubernetes.io/docs/concepts/workloads/autoscaling/horizontal-pod-autoscale/

Kumar, J., & Singh, M. (2025). *System Design on AWS* (First Edition ed.). O'Reilly Media, Inc.

Linthicum, D. (2024, 2). *Why companies are leaving the cloud.* Retrieved from https://www.infoworld.com/article/3712861/why-companies-are-leaving-the-cloud.html

Maarek, S. (2019, 1). *gRPC [Java] Master Class: Build Modern API and Microservices.* Retrieved from https://learning.oreilly.com/course/grpc-java-master/9781838558048/

MACH Alliance. (2023a). *THE NEW APPROACH TCO e-Book to TCO for digital commerce.* commercetools, Google Cloud. Retrieved from https://machalliance.org/insights-hub/

the-new-approach-to-tco-for-digital-commerce

MACH Alliance. (2023b, 5). *The New Approach to TCO for Digital Commerce.* Retrieved from https://machalliance.org/insights-hub/the-new-approach-to-tco-for-digital-commerce

MACH Architecture. (2021a). *What are the Advantages of MACH Architecture?* Retrieved from https://macharchitecture.com/articles/what-are-the-advantages-of-mach-architecture

MACH Architecture. (2021b). *What is MACH Architecture?* Retrieved from https://macharchitecture.com/articles/what-is-mach-architecture

Martin, R. C. (2003, 5). *UML FOR JAVA™ PROGRAMMERS.* Retrieved from https://learning.oreilly.com/library/view/uml-for-javatm/0131428489/

Mastrangelo, C. (2018, 12). *Visualizing gRPC Language Stacks.* Retrieved from https://grpc.io/blog/grpc-stacks/

McGrew, D. (2008, 1). *RFC 5116: An Interface and Algorithms for Authenticated Encryption | RFC Editor.* Retrieved from https://www.rfc-editor.org/info/rfc5116/

Mezzalira, L. (2025, 10). *Building Micro-Frontends, 2nd Edition.* Retrieved from https://learning.oreilly.com/library/view/building-micro-frontends-2nd/9781098170776/

Montalion, D. (2024, 7). *Learning Systems Thinking.* Retrieved from https://learning.oreilly.com/library/view/learning-systems-thinking/9781098151324/

Moorhead, P. (2020, 5). *AWS, COVID-19, And The Need For Speed In Time Of Crisis.* Retrieved from https://www.forbes.com/sites/moorinsights/2020/05/20/aws-covid-19-and-the-need-for-speed-in-time-of-crisis/?sh=422c2d6d7ad8

Morris, K. (2020). *Infrastructure as Code, 2nd Edition.* O'Reilly Media, Inc. Retrieved from https://www.oreilly.com/library/view/infrastructure-as-code/9781098114664/

MTF Institute of Management, T., & Finance. (2024, 9). *Professional Diploma in Unit Economics Management.* Retrieved from https://www.udemy.com/course/professional-diploma-in-unit-economics-and-sales-funnels

MvnRepository. (2026, 2). *Maven Repository: software.amazon.awssdk » pricing » 2.41.34.* Retrieved from https://mvnrepository.com/artifact/software.amazon.awssdk/pricing/2.41.34

Newman, S. (2015). *Sam Newman Principles Of Microservices.* Retrieved from https://samnewman.io/talks/principles-of-microservices/

Newman, S. (2019). *Monolith to Microservices* (1st Edition ed.). O'Reilly.

Newman, S. (2021). *Building Microservices* (Second Edition ed.).

Newton-King, J. (2022, 11). *G2: gRPC over HTTP/3.* Retrieved from https://github.com/grpc/proposal/blob/master/G2-http3-protocol.md

Olsen, R. (2024, 9). *To The Moon.* Retrieved from https://www.youtube.com/watch?v=ntmkMLcveC0

O'Reilly. (2019, 1). *gRPC [Java] Master Class: Build Modern API and Microservices.* Retrieved from https://learning.oreilly.com/course/grpc-java-master/9781838558048/

O'Reilly, B. (2024, 6). *Residues: Time, Change, and Uncertainty in Software Architecture.* Retrieved from https://leanpub.com/residuality

Pascal, E. (1895). *Esercizi E Note Critiche Di Calcolo Infinitesimale.* Legare Street Press.

Perkins, B., & Panek, W. (2021). *Microsoft Azure Architect Technologies and Design Complete Study Guide EXAMS AZ-303 AD AZ-304.* Sybex a Wiley Brand.

Peterson, J. (2019). *12 Rules for Life: An Antidote to Chaos.* Penguin Random House UK. Retrieved from https://a.co/d/0j0t23Wb

QConPlus 2021: Takeout burritos and minimizing design-time coupling in a microservice architecture. (2021). Retrieved from https://microservices.io/post/microservices/2021/05/21/qcon-2021-loose-design-time-coupling.html

Raschka, S. (2025). *Build a Large Language Model (From Scratch).* Retrieved from https://learning.oreilly.com/library/view/build-a-large/9781633437166/

Ray, M. (2023, 2). *OpenCost is a FinOps Certified Solution! | OpenCost — open source cost monitoring for cloud native environments.* Retrieved from https://opencost.io/blog/finops-certified-solution/

Read, J. (2023, 10). *Communication Patterns.* Retrieved from https://learning.oreilly.com/library/view/communication-patterns/9781098140533/

Rescorla, E. (2026, 7). The Transport Layer Security (TLS) Protocol Version 1.3. Retrieved from https://www.rfc-editor.org/info/rfc9846 doi: 10.17487/RFC9846

Rescorla, E., & Rescorla, E. (2018, 8). The Transport Layer Security (TLS) Protocol Version 1.3. Retrieved from https://www.rfc-editor.org/info/rfc8446/ doi: 10.17487/RFC8446

Richards, M. (2024, 2). *(461) Lesson 181 - Feasibility and Questioning Requirements - YouTube.* Retrieved from https://www.youtube.com/watch?v=65qOqfbQV2Y

Richards, M., Lange, B., & Ford, N. (2021). *Fundamentals of Software Architecture: An Engineering Approach.*

Richardson, C. (2019). *Microservices Patterns.* MANNING.

Richardson, C. (2020). *Cubes, Hexagons, Triangles, and More: Understanding Microservices by Chris Richardson - YouTube.* Retrieved from https://www.youtube.com/watch?v=rMDjuXTQVkk

San Miguel, D., Alfonso; Obando. (2024). *Efficient Cloud FinOps.* Packt Publishing Ltd.

Schwarz, K., Moran, J., & Bachmeier, N. (2024). *Engineering Resilient Systems on AWS* (First Edition ed.). O'Reilly.

Serao, M. (2025). *El vientre de Nápoles.* Gallo Nero Ediciones, S. L.

Smart, J. F., & Molak, J. (2023). *BDD in Action* (Second Edition ed.). MANNING.

Snowden, D. (2005). *The Cynefin Framework.* Retrieved from https://thecynefin.co/about-us/about-cynefin-framework/

Solomonson, C. (2024, 5). *The Composable Roadmap: An action plan for agility in a modern digital marketplace.* Retrieved from https://www.amazon.com/Composable-Roadmap-agility-digital-marketplace/dp/B0D3GXVDC8

Stevens, D. (2025). *Mach Architecture: Microservices, API-first, Cloud-native, and Headless principles* (Edición Kindle ed.). Gareth & Co - Publisher House.

Storment, J., & Fuller, M. (2023). *Cloud FinOps* (Second ed.). O'Reilly Media, Inc.

The Apache Software Foundation. (2012). *Apache Mesos.* Retrieved from https://mesos.apache.org/

The Kubernetes Authors. (2026). *Kubernetes.* Retrieved from https://kubernetes.io/

The Linux Foundation. (2024). *FinOps for Containers.* Retrieved from https://learn.finops.org/finops-for-containers

Thymeleaf. (2026, 4). *Tutorial: Thymeleaf + Spring.* Retrieved from https://www.thymeleaf.org/doc/tutorials/3.1/thymeleafspring.html

Trifork. (2024). *2 Days: Visualising Software Architecture with the C4 Model.* Retrieved from https://gotocph.com/2024/masterclasses/439/2-days-visualising-software-architecture-with-the-c4-model

Trivedi, S. (2023, 9). *Five things that differentiate cloud cost management and FinOps.* Retrieved from https://www.thoughtworks.com/insights/blog/cloud/5_cloud_cost_management_finops

U.S. Small Business Administration. (2024, 10). *Break-even point | U.S. Small Business Administration.* Retrieved from https://www.sba.gov/business-guide/plan-your-business/calculate-your-startup-costs/break-even-point

Vanner, C. (2024, 8). *A Quick Guide to BPMN: Business Process Model and Notation.* Retrieved from https://www.bizagi.com/en/blog/guide-to-bpmn

Veneranda Fabrica del Duomo di Milano. (2023). *Duomo Catedrale di Milano* (NOUS Srl, Ed.). NOUS Srl.

Visual Paradigm. (2026). *Visual Paradigm Online.* Retrieved from https://online.visual-paradigm.com/w/nlippcxu/diagrams/#diagram:proj=0&type=SequenceDiagram&width=11&height=8.5&unit=inch

Vogels, W. (2023a). *AWS re:Invent 2023 - Keynote with Dr. Werner Vogels.* Retrieved from https://www.youtube.com/watch?v=UTRBVPvzt9w

Vogels, W. (2023b). *THE FRUGAL ARCHITECT.* Retrieved from https://www.thefrugalarchitect.com/

Wynne, M. (2015, 12). *Introducing Example Mapping | Cucumber.* Retrieved from https://cucumber.io/blog/bdd/example-mapping-introduction/